Janne Salovaara

Esittelyversion luominen kunnonvalvonnan mittalaitteistosta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

6.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Janne Salovaara Esittelyversion luominen kunnonvalvonnan mittalaitteesta 43 sivua + 3 liitettä 6.4.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine	Automaatiotekniikka
Ohjaajat	Diplomi-insinööri Kari Aittanen Lehtori Timo Kasurinen

Työn tarkoituksena oli luoda yritykselle esittelyversio kunnonvalvontalaitteistosta, joka pystyisi mittaamaan esimerkiksi lämpötilaa, tärinää ja koneen käyntiaikoja. Näiden arvojen avulla pystytään ennustamaan mahdollisten vikojen syntymiset, jos ne kasvavat vertailuarvoihin verrattuna liian korkeiksi. Työtä varten perehdyttiin kunnossapitoon yleisesti sekä kunnonvalvontaan, jossa erityisesti tutkittiin, kuinka mitatun tiedon avulla voidaan ennustaa laitteen vikaantumisia.

Kunnonvalvontalaitteiston alustaksi valikoitui Raspberry Pi -pienoistietokone, jonka digitaalisten sisääntulojen ja Ethernet-verkkoliittimen avulla pystytään mittaamaan ja siirtämään tietoa tietokantaan. Tiedonsiirtoa varten valittiin vaihtoehdoiksi kaksi erilaista yhteyskäytäntöä: MQTT ja OPC UA. Lopulliseksi yhteyskäytännöksi tiedonsiirtoon valikoitui OPC UA -yhteyskäytäntö yrityksen valmiiden tietojärjestelmien takia. OPC UA -palvelin asennettiin Raspberry Pi -tietokoneelle, josta antureilta saatu tieto siirrettiin eteenpäin tietokannalle. Työssä tarvittiin myös KepwareServerEx-ohjelmistoa, jonka avulla tieto muutettiin tietokannalle sopivaksi OPC DA -yhteyskäytännöksi.

Lopullinen esittelyversio pystyi kirjoittamaan tietokantaan haluttuja arvoja, mutta versio vaatii vielä lisäkehitystä, jotta siitä pystyttäisiin rakentamaan tuotantokäyttöön sopiva laitteisto. Työssä esitellään myös vaihtoehtoisia ratkaisuja Raspberry Pi -tietokonetta käyttäen niin, että mittaustietoa ei lähetetä tietoverkossa reaaliaikaisesti tai se siirretään käyttäen langattomia yhteyksiä.

Avainsanat	Kunnonvalvonta, OPC, MQTT, Raspberry Pi, kunnossapito

Author	Janne Salovaara
Title	Proof of Concept on a Predictive Maintenance Data Collector
Number of Pages	43 pages + 3 appendices
Date	6 April 2018
Degree	Bachelor of Engineering
Degree Programme	Electrical and automation engineering
Professional Major	Automation Technology
Instructors	Kari Aittanen, Master of Science in Technology Timo Kasurinen, Senior Lecturer

The object of this final year project was to create a proof of concept for a company on a predictive maintenance system that could measure for example temperature, vibration and operating time. With these measurements it is possible to predict future faults by comparing collected data to the historical mean. If the measurements rise above a certain threshold, the machine should be investigated for possible problems. This thesis also presents issues on maintenance, predictive maintenance and how collected data can be used to determine when failure is going to occur.

Raspberry Pi minicomputer was chosen as the platform for this predictive maintenance system, as it has digital inputs and an Ethernet connection which are used to collect data from the sensors and transmit it to the database. Two protocols were tested in this project for data transmission: MQTT and OPC UA. Final proof of concept product uses OPC UA because the company that requested this thesis uses systems that already support this protocol. The OPC UA server was installed on Raspberry Pi that communicated with KepwareServerEx program which bridged OPC UA and database together using OPC DA protocol.

The final product could write temperature, acceleration and operating time data to the database, but to use this product in production still needs extra development. Optional versions of the product could include wireless connection or store the data locally.

Keywords	Predictive maintenance, OPC, MQTT, Raspberry Pi, maintenance

Sisällys

Liitteet

Liite 1. OPC UA -palvelimen Python-ohjelma

Liite 2. MQTT-tiedonsiirtoon Python-ohjelma

Liite 3. Kiihtyvyysanturin ADXL345 C-ohjelma SPI-yhteyskäytännöllä lukemiseen

Lyhenteet ja käsitteet

AD-muunnin	Muuntaja, joka muuttaa analogisen signaalin digitaaliseksi.
MQTT	Message Queuing Telemetry Transport, kevyt standardoitu tiedonsiirtoyhteyskäytäntö esineiden internetin ratkaisuihin.
OPC	Standardoitu tiedonsiirtoyhteyskäytäntö automaatiosovellusten käyttöön.
OPC DA	Klassisen OPC:n tiedonsiirtomenetelmä arvojen lukemiseen ja kirjoittamiseen.
OPC UA	OPC Unified Architecture, yhteyskäytäntö automaation tiedonsiirtoon järjestelmien välillä.
PoE	Power over Ethernet, tekniikka, jossa on samassa kaapelissa virransyöttö ja Ethernet-lähitietoverkkoyhteys.
Standardi	Virallisen organisaation esittämä määrittely, miten standardin määrittelemät asiat tulee tehdä.
TCP/IP	Transmission Control Protocol / Internet Protocol, internetliikennöintiin käytetty tietoverkkoyhteyskäytäntö.
WLAN	Wireless Local Area Network, langaton lähitietoverkkotekniikka.

1 Johdanto

Yritysten halutessa vähentää kunnossapidon kustannuksia ja kasvattaa tuotantolaitteittensa käytettävyyttä on kasvanut tarve menetelmille, joilla pystytään tukemaan nykyaikaisen kunnossapidon toimenpiteitä. Kunnonvalvonta on yksi näistä toimenpiteistä. Kunnonvalvonta tarkoittaa järjestelmää, joka mittaa laitteen kuntoa vertaamalla sitä kerättyyn historialliseen tietoon.

Insinöörityön tarkoituksena on esitellä yritykselle toteutettua esittelyversiota tiedonkeruulaitteesta, jolla voitaisiin tehostaa yrityksen tuotantolaitteiden kunnonvalvontaa. Tiedonkeruulaite yhdistettäisiin yrityksen jo valmiisiin tietokantoihin, millä vähennetään ylläpitoresursseja sekä hyödynnetään valmista osaamista. Laitteen tuli olla kustannustehokas, monistettavissa ja rakennettavissa kaupallisista komponenteista. Yrityksen pyrkimyksenä erityisesti oli lisätä kunnonvalvontaa laitteille, joilla ei ole kunnonvalvontaan kykenevää automaatiojärjestelmä. Laitteen siis tulisi olla yksinkertaisesti asennettavissa tuotantokoneiden yhteyteen niin, että mittaukset ja tiedonsiirto voidaan suorittaa suoraan yhdellä laitteella. Laitteella haluttaisiin mitata koneiden kuntoa siten, että laite mittaisi parametreja valvottavalta laitteelta kuten lämpötilaa, koneen tärinää ja koneen käyntiaikoja. Vertaamalla mittausten historiatietoja voidaan ajan kuluessa huomata kasvavat trendit, mahdolliset vikaantumiset ja huoltaa laitteita ennen niiden täydellistä hajoamista.

Insinöörityössä esitellään kunnossapitoa osana automaatiota ja kunnonvalvontalaitteen suunnittelua, rakennetta, käyttökohteita ja sen toimintaa. Tämän lisäksi työssä esitellään eri mahdollisuuksia luoda yhteys yrityksen valmiiden järjestelmien ja mittauslaitteen välillä niin, että mittalaitteistossa on omaa älyä ja laite pystyy kommunikoimaan itsenäisesti käytössä olevien järjestelmien kanssa. Anturitekniikkaan ja mekaniikkaan ei työssä perehdytä testeissä käytettyjen antureiden lisäksi syvemmin.

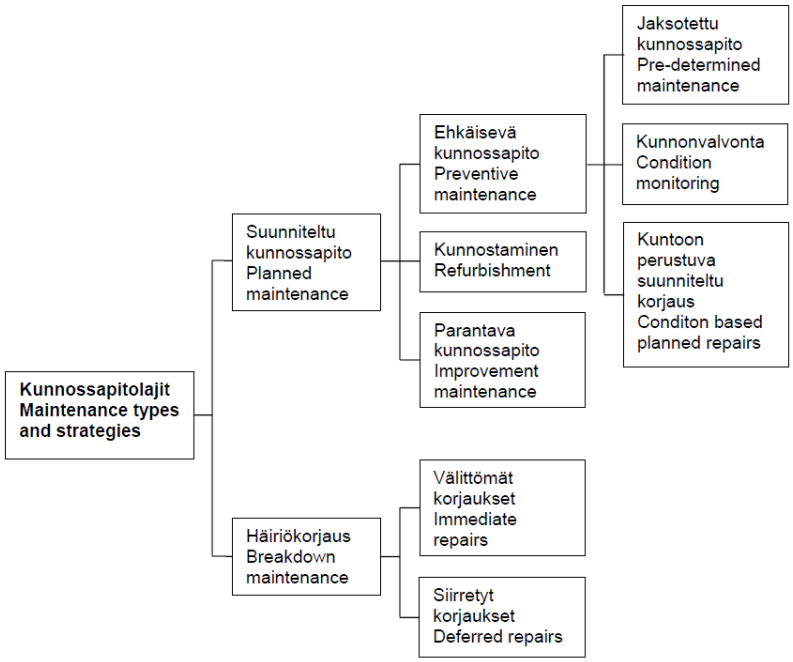
2 Kunnossapito

2.1 Kunnossapidon määritelmä

Kunnossapito tarkoittaa niitä toimenpiteitä, joiden avulla pidetään kohteet tehokkaassa toimintakunnossa mahdollisimman pitkänaikaa. PSK6201-standardin mukainen määritelmä kunnossapidolle on seuraava:

> Kunnossapito on kaikkien niiden teknisten, hallinnollisten ja johtamiseen liittyvien toimenpiteiden kokonaisuus, joiden tarkoituksena on säilyttää kohde tilassa tai palauttaa se tilaan, jossa se pystyy suorittamaan vaaditun toiminnon sen koko-elinjakson aikana [1, s. 2].

Tuotannossa yhä entistä tärkeämpää on laitteiden käytettävyys, joka tarkoittaa kohteen kykyä toimia halutussa tilassa ja tietyissä olosuhteissa tietyllä aikavälillä, kun tarvittavat resurssit ovat käytettävissä [1, s. 5]. Jotta käytettävyys olisi mahdollisimman korkea, on suoritettava suunnitelmallisia kunnossapidon toimenpiteitä. Kunnossapidolle pitäisi siis luoda selkeä strategia, jonka avulla voidaan kattaa yrityksen kunnossapitotarpeet niin, että kunnossapidon muodot ovat mahdollisimman monipuoliset. [1, s. 13.]



Kuva 1. Kunnossapito lajitellaan kahteen haaraan PSK6201-standardin mukaan [1, s. 21].

Kunnossapito yleisesti koostuu kahdenlaisesta kunnossapidon lajista: suunnitellusta ja korjaavasta kunnossapidosta. Kunnossapidon eri lajit on havainnollistettu kuvassa 1, jossa suunnitellun kunnossapidon alla ovat häiriöitä ehkäisevät toimenpiteet ja häiriökorjauksen alla välittömät ja siirretyt korjaukset. Korjaava kunnossapito pitää sisällään vian havaitsemisen jälkeen tehtävät toimenpiteet. Kun vika on havaittu, vika voidaan korjata heti tai tehdä kunnossapitotoimet siirretysti esimerkiksi tuotantopysähdyksen aikana [1, s. 23]. Ennalta suunnittelematonta korjaavaa kunnossapitoa pitäisi välttää mahdollisimman paljon, sillä se tarkoittaa usein sitä, että tuotanto joudutaan keskeyttämään suunnitelmattomasti laiterikon vuoksi. Yksittäisten vikojen korjaaminen välittömästi voi aiheuttaa seuraavia kustannuksia ja menetettyjä tuottoja:

- tuotannon menetys
- laadun heikkeneminen
- tyytymätön asiakas
- organisaation tehottomuus
- korkeammat korjauskustannukset äkillisyydestä johtuen.

Pyrkimys kunnossapidossa on, että suurin osa vioista pystytään ennustamaan tai ehkäisemään. Kuitenkin yllätyksellisiä laiterikkoja esiintyy aika ajoin, jolloin niiden priorisointi on tärkeää ja tarvittavat resurssit ovat oltava vian korjaamiseen. [1, s. 24.]

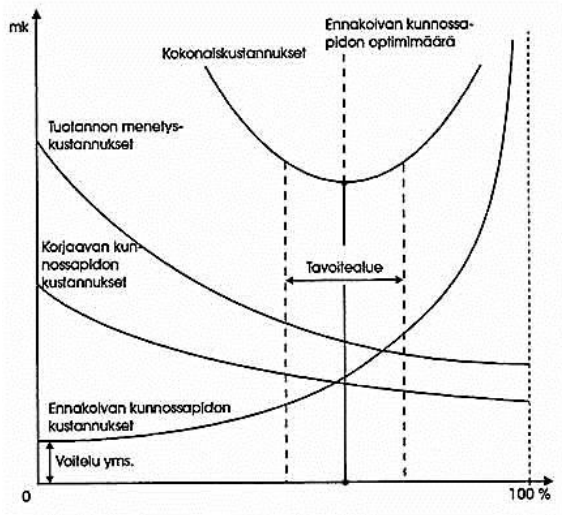
Ennen vian havaitsemista on mahdollista tehdä ehkäisevää kunnossapitoa. Tämä perustuu joko mitattuun tai suunniteltuun kunnossapitoon, jolloin vika yritetään estää tai sen vaikutus minimoidaan. Jaksollisesti tehtävät kunnossapitotoimet tehdään suunnitellusti aikajaksoittain, millä vähennetään vikoja eri toimenpiteiden avulla. Tällaisia kunnossapitotehtäviä ovat esimerkiksi ajoitetut rasvaukset ja puhdistukset. Toinen ehkäisevän kunnossapidon laji on kunnonvalvonta. Kunnonvalvonta perustuu mittatietoon, jonka avulla voidaan ennustaa tulevat viat, tehdä kunnossapitotoimet suunnitellusti ja välttää turhat kunnossapitotoimet. [2, s. 3.]

2.2 Kunnonvalvonta

Kunnossapitoon käytetään noin 15–60 % tuotteen valmistuskuluista, joten on tärkeää mitoittaa kunnossapitoon käytettävät resurssit oikein ja vähentää turhien toimenpiteiden määrää [3, s. 1]. Ehkäisevällä kunnossapidolla pyritään pidentämään laitteen käyt-

töikää ja vähentämään yllätyksellisten vikojen esiintymistä. Verrattuna kunnossapitofilosofiaan, jossa ajetaan laiteet rikkoon saakka ilman ehkäiseviä kunnossapitotoimenpiteitä, vähentää ehkäisevä kunnossapito jopa kolminkertaisesti kunnossapidon kuluja [3, s. 3]. Kunnossapidon korjaavia kustannuksia voidaan vähentää käyttämällä ehkäisevää kunnossapitoa oikealla tavalla, käyttäen kunnonvalvontaa ja suunniteltuja kunnossapitotoimia.

Ehkäisevässä kunnossapidossa on kuitenkin tasapainotettava kokonaiskustannukset niin, ettei ylimääräistä ehkäisevää kunnossapitoa toteuteta turhaan. Ehkäisevällä kunnossapidolla pystytään vähentämään korjaavan kunnossapidon kustannuksia, kunnes saavutetaan piste, jossa sillä saavutetut kustannusvähennykset eivät kata siihen kulutettuja resursseja. Kuvassa 2 on kuvattu tavoitealuetta, jossa ennakoivan kunnossapidon kustannukset eivät ole kasvaneet vielä liian korkeiksi, mutta korjaavat kunnossapidon kustannukset ja tuotannon menetyksistä aiheutuvat kustannukset on minimoitu samalla. [4.]



Kuva 2. Ennakoivan kunnossapidon kustannukset pitää tasapainottaa kokonaiskustannuksiin verrattuna [4].

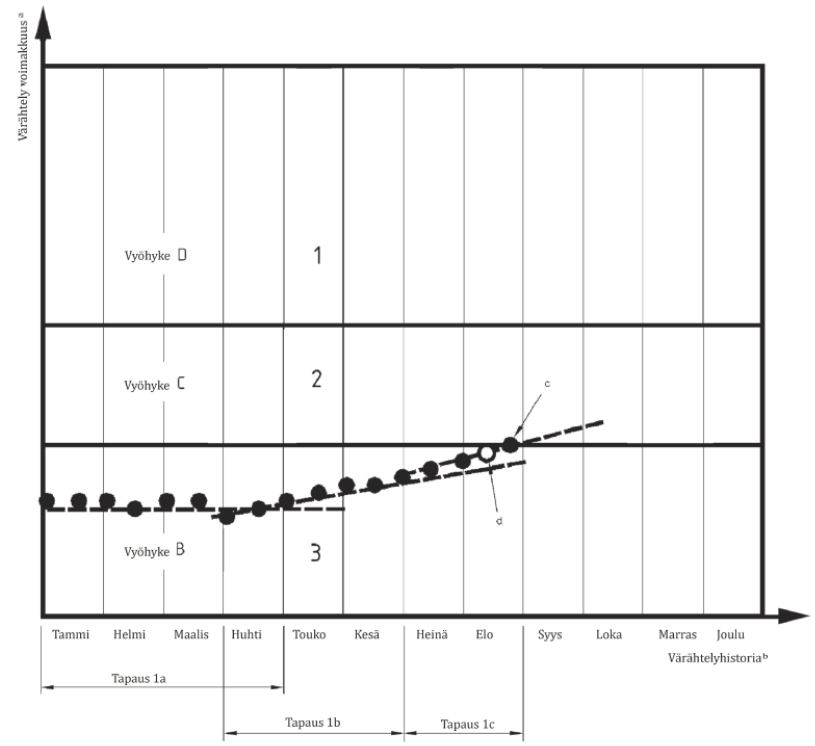
Kunnonvalvonnan avulla pyritään havaitsemaan alkavat vikaantumiset mittaamalla laitteesta suureita ja tarkastelemalla poikkeavatko ne normaalista. Kun suureet poikkeavat normaalista tarpeeksi, kunnonvalvontajärjestelmän pitää hälyttää mahdollisesta tulevasta laiteviasta. Tällaisen järjestelmän avulla voidaan havaita mahdolliset ongelmat, jonka jälkeen täytyy selvittää ongelman syy, arvioida sen vakavuus ja suunnitella korjaustoimenpiteet. [2, s. 4.]

Kunnonvalvonnan tavoitteena on myös poistaa joitakin turhia kunnossapitotoimia, joita tehdään määräaikaisesti laitteen huoltamiseksi. Tällöin laitetta voidaan käyttää pidempään ja paremmin ilman huoltotoimenpiteitä, joita esimerkiksi ovat laakereiden vaihdot. Aiemmin jopa 33 % kunnossapitoon käytetyistä varoista kulutettiin turhiin kunnossapitotoimiin, joita voidaan vähentää oikeanlaisilla kunnonvalvontajärjestelmillä. [3, s. 1.]

Kunnonvalvonnan yleisempiä mittasuureita ovat

- lämpötila
- tärinä
- voiteluöljyn puhtaus
- sähkövirta
- muut prosessisuureet kuten paine, virtaus ja käyntinopeus [2, s.4].

Tärinä on kunnonvalvonnan yksi tärkeimmistä mittauksista, ja sen avulla voidaan mitata esimerkiksi moottorin laakereiden kuntoa. Voitelun ja laakereiden heiketessä kitkavoimat liikkuvissa osissa kasvavat, mikä aiheuttaa ylimääräistä tärinää moottorin osiin, jolloin moottorin värähtelytaajuus muuttuu ja moottorin liikkeestä johtuvat rikkoutumiset lisääntyvät. Värähtelyä voidaan mitata käyttäen kiihtyvyyttä, nopeutta tai poikkeumaa lepopisteestä. Värähtelyn mittauksessa on tärkeää löytää anturille oikea kohta laitteesta sekä kiinnittää mittaava anturi mahdollisimman tiukasti mitattavaan laitteeseen. Kuvassa 3 on kuvattu kuinka värähtelyn voimakkuuden kasvusta voidaan päätellä, milloin laitteen kunto on alkanut heiketä, kun voimakkuus kasvaa värähtelyrajojen yli. [2, s. 4–8.]



Selite

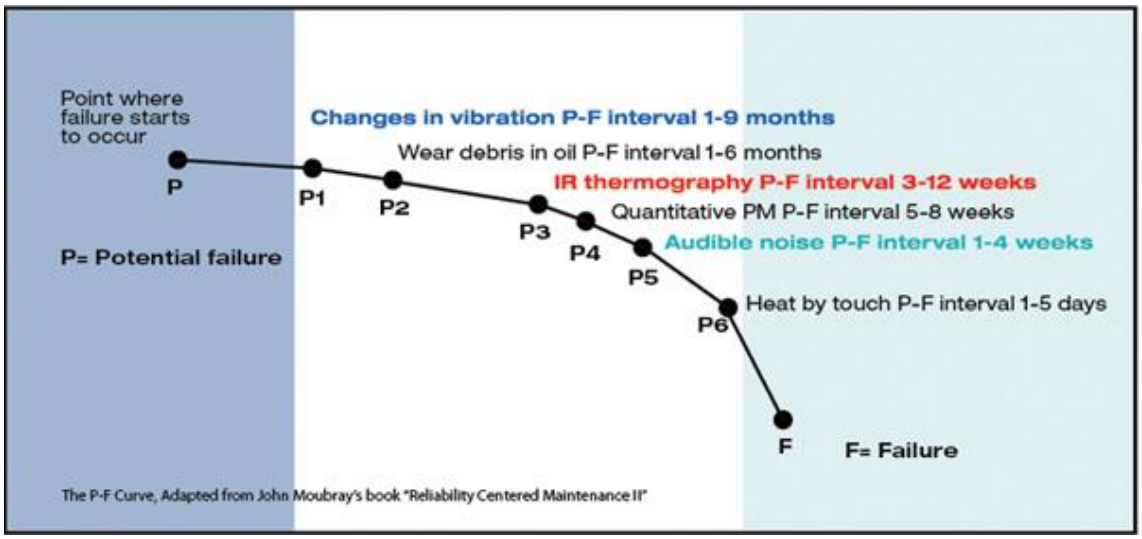
1 Toimenpiteitä tarvitaan
2 Hälytys
3 Normaali

a Huipusta huippuun siirtymä (m) tai rms nopeus (mm/s)
b Katso värähtelyrajat ISO 7919 tai ISO 10816
c Projisoitu tarvittavien toimenpiteiden ajankohta
d Seurantaa vaaditaan 48 tunnin sisällä

Kuva 3. Mitatun tärinän historiatiedosta voidaan päätellä, milloin laitteen kunto alkaa heiketä ja milloin toimenpiteitä vaaditaan [5, s. 30].

Tärinämittauksille on kolmen tyyppisiä järjestelmiä: kiinteästi asennetut, puolikiinteästi asennetut ja kannettavat valvontajärjestelmät. Kiinteästi asennetuilla järjestelmillä mittaus, tallennus ja tiedonkäsittely on kiinteästi asennettuna mitattavaan laitteistoon ja tietoa siirretään jatkuvasti tai jaksottaisesti tietoverkossa. Kannettavassa järjestelmässä valvontalaite on siirrettävissä kokonaisuudessaan mitattavaan kohteeseen, jossa järjestelmä voi kerätä tietoa ennalta määritetyltä, esimerkiksi päivän mittaiselta, ajanjaksolta. Tämä tieto voidaan siirtää samalla tavalla kuin kiinteässäkin järjestelmässä tietoverkon yli tai se voidaan tallentaa paikallisesti valvontalaitteen muistiin, josta se voidaan myöhemmin siirtää analysoitavaksi toiseen järjestelmään, kun laite on poistettu kohteesta. Puolikiinteäjärjestelmä on kannettavan ja kiinteän järjestelmän välimuoto, jossa anturit ovat kiinteästi mitattavassa laitteessa, mutta mittalaite on siirrettävissä. [5, s. 7.]

Lämpötilamittauksen avulla voidaan mitata laitteen vioittumisesta aiheutuvia lämpötilamuutoksia. Laitteen vikaantuessa lämpötila kasvaa kohteen ympäristössä, mikä voidaan havaita vertaamalla normaalitilassa olevan laitteen lämpötilaa mitattuihin arvoihin. Lämpötilan kasvaessa yli laitteelle määritetyn raja-arvon kunnonvalvontajärjestelmän tulisi ilmoittaa mahdollisesti vikaantumisesta. Lämpötilan kasvamiseen syynä voi olla esimerkiksi lisääntynyt kitka tai sähkötekniset viat. Esimerkiksi moottorin lämpötilan kasvaessa voi moottorin voitelu olla liian vähäistä, jolloin lämpöä tuottavat kitkavoimat kasvavat. Tärkeää lämpötilan mittauksessa on paikka, josta sitä mitataan. Jos lämpötilaa mitataan liian kaukaa laitteen kriittisistä osista, jotka lämpenevät vioittuessaan, vika huomataan liian myöhään. Tärinämittaukseen verrattuna lämpötilamittauksella kuitenkin usein huomataan viat myöhemmässä vaiheessa, jolloin laite on jo voinut vioittua ja kunnossapitotoimia ei ehditä tehdä ajoissa. Tällöin on tärkeää, että mittaustieto saadaan mahdollisimman nopeasti analysoitua ja hälytys viasta lähetettyä huolto-osastolle. Kuvan 4 mukaisesti voidaan huomata kuinka tärinän avulla voidaan alkaa havaita vikoja huomattavasti aikaisemmin kuin lämpötilan avulla. [2, s. 5.]



Kuva 4. Laitteen kunnon heiketessä alkavat viat voidaan havaita eri tavoin eri vaiheissa [6].

Ehkäisevään kunnossapitoon voidaan käyttää myös hyväksi laitteiden käyntitietoa laskemaan, milloin laitteelle tulisi tehdä väliaikaisia huoltotoimenpiteitä. Käyntitiedon avulla saadaan myös suodatettua sellaista tietoa pois kunnonvalvonnan mittauksista, milloin laite ei ole ollut käytössä. Tällöin on helpompaa verrata laitteesta mitattuja arvoja aikaisempiin arvoihin, milloin laite on oikeasti ollut käynnissä.

3 Laitevaatimukset

Kunnonvalvontalaitteelle muodostui erilaisia vaatimuksia sen toiminnasta mittalaitteena, sen yhteyskäytäntöjen mahdollisuuksista ja yrityksen asettamista raja-arvoista. Laitteen piti kyetä mittaamaan ainakin lämpötilaa, tärinää sekä laitteen käyntiaikaa. Tätä varten laitteen tuli pystyä käyttämään hyväksi niin analogisia kuin digitaalisia signaaleita. Tärinä ja lämpötila ovat tärkeitä mittareita laitteen kunnon arviointiin, ja käyntiaikaa voidaan käyttää ajanjaksollisten kunnossapitotoimenpiteiden suunnitteluun [2, s. 4]. Laitteen tuli olla myös riittävän kyvykäs tiedonsiirtoon ja tärinämittausten analysointiin, sillä laitteesta haluttiin yksittäinen koottu paketti. Lisäksi tärinämittaukseen tuli löytää ratkaisu, jossa tärinämittaus saataisiin analysoitua mittauslaitteella. Tämä johtui tarpeesta mitata tärinää kiihtyvyyden avulla, jolloin mittauslaitteen pitäisi mitata kiihtyvyyttä tuhansia kertoja sekunnissa. Tietoverkossa näin monen mittaustiedon siirtäminen näin nopeasti aiheuttaisi tietoverkkoon tukoksia ja hidastumista.

Tiedonsiirtoa varten käytettävien yhteyskäytäntöjen, eli eri tahojen määrittelemät säännöt viestinnälle, tuli kyetä lähettämään mitattuja arvoja Ethernet-väylän ylitse. Yrityksellä on jo käytössään valmiita tietoverkkolaitteita, ja jotta viestintä tietokannalle toimii, täytyy sen kulkea yrityksen sisäverkon lävitse Ethernet-kaapelilla TCP/IP-yhteyskäytäntöä käyttäen. Lisäksi yhteyskäytäntöjen täytyi pystyä toimimaan yrityksen valmiiden ohjelmistojen kanssa. Työn tilanneen yrityksen tietokanta vaatii OPC DA -yhteyskäytäntöä tiedon lukemiseen mittalaitteistoilta. OPC DA -viestintä taas vaatii Windows-ympäristöä ja se ei kykene helposti siirtämään tietoa palomuurien läpi, joten viestintään tarvitaan toinen viestintämuoto, jolla tieto voidaan siirtää ensin Windows-tietokoneelle. Tällä tietokoneella voidaan tieto muuttaa OPC DA -yhteyskäytännölle sopivaksi, jotta yrityksen tietokannan lukija-ajurit voivat lukea mittaustiedot. Yritys on näiden lisäksi antanut ehdotuksia laitteen toiminnasta. Laitteen tulisi olla edullinen, monistettavissa, pystyä rakentamaan kaupallisista osista sekä toimia yrityksen valmiiden järjestelmien kanssa.

Kunnonvalvonnan kannalta mittauksella tuli pystyä määrittämään laitteen nykykunto, joka voidaan tallentaa historiatietoihin. Historiatietojen perusteella pitäisi pystyä vertaamaan mitattavan laitteen kunnonkehittymistä siten, että laitteelle voidaan tehdä huoltotoimenpiteitä, kun laitteen mittausarvot poikkeavat tarpeeksi hyväkuntoisen laitteen arvoista. [7, s. 35.]

4 Järjestelmävaihtoehdot

Laitteen haluttiin olla helposti muokattavissa, edullinen ja pystyä mittamaan haluttuja arvoja, mikä vaikutti järjestelmävaihtoehtoihin. Valmiit teollisuuden internet laitteet, automaation logiikat ja älykkäät I/O-laitteet todettiin hinnaltaan liian korkeiksi siihen, että laitteita haluttaisiin monistaa ympäri tehtaita. Esimerkiksi Raspberry Pi -mittauslaitteen osat maksavat alle 100 euroa, jolloin laitteen monistaminen on halpaa kaluston osalta. Laiteelta vaadittiin myös älyä tärinämittaukseen, koska sen mittaustaajuus tulisi kattaa mitattavien kohteiden värähtelytaajuudet, jotka yleisesti ovat kokoluokaltaan 0,1–30 kHz [5, s. 21]. Tällöin laitteen tulisi mitata esimerkiksi kiihtyvyyttä tuhansia kertoja sekunnissa, jotta tärinän kehittymistä pystytään edes jossain määrin tutkimaan. Tästä syystä tietoa kerääntyy paljon ja sitä ei voida suoraan tietoverkossa siirtää, joten sitä on analysoitava suoraan mittauslaitteella. Esimerkiksi Älykkään I/O-laitteen ioLogik2240:n avulla olisi mahdollista mitata suureita käyttäen analogisia antureita, mutta logiikalla ei ole mahdollista toteuttaa suoraa tärinämittausta sen matalasta mittaustaajuudesta johtuen ja sen kyvyttömyydestä analysoida tietoa [8]. Kunnonvalvonnan kannalta tämä haittaa tarkan kuvan saamista laitteen nykykunnosta.

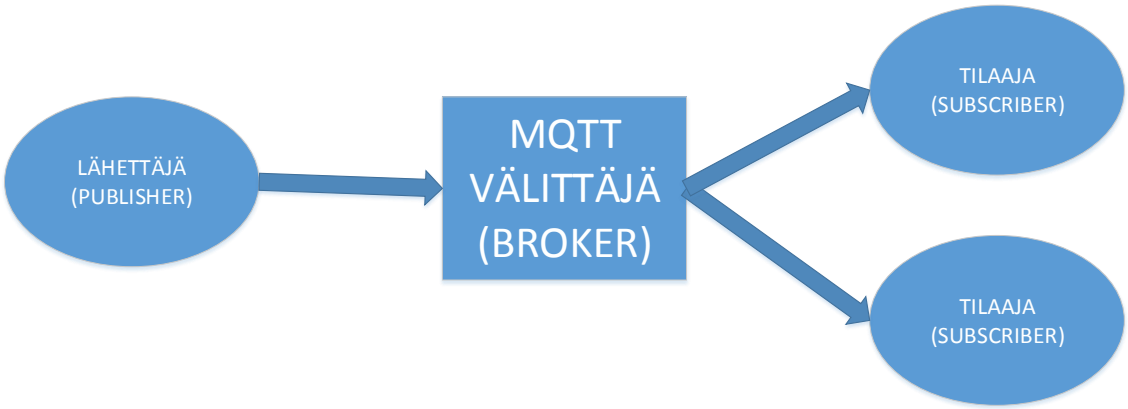
Tiedon prosessointiin ja tiedonsiirtoon tarvittiin riittävästä tehoa, jolloin vaihtoehdoiksi valikoitui Raspberry Pi ja Arduino-mikrokontrollerilevyt. Raspberry Pi ja Arduino ovat minitietokoneita, jotka toimivat samantyyppisillä prosessoreilla kuin nykyiset matkapuhelimet. Muistia laitteissa on versiosta riippuen yhteen gigatavuun saakka. Tämä riittää sopivasti pyörittämään viestintä- ja mittausohjelmia, joihin voidaan lisätä älyä ohjelmoimalla sopivia ohjelmia. Raspberry Pi sisältää digitaalisia sisääntuloja antureita varten, ja Arduino sisältää näiden lisäksi analogisia sisääntuloja. Nämä laitteet voisivat toimia raskaampien järjestelmien tukena mittaamassa yksittäisiä kohteita, joihin ei kokonaista teollisuustason kunnonvalvontajärjestelmää haluta asentaa.

Tiedonsiirtomahdollisuuksina kokeiltiin MQTT- ja OPC UA -yhteyskäytäntöjä. MQTT on hyvin kevyt tapa viestiä, ja se sopii nykyaikaisiin suuria määriä tietoa lähettäviin teollisen internetin ratkaisuihin, mutta vaatii nykyaikaisia ajureita, joita ei vanhoissa tuotantoympäristöissä välttämättä vielä käytetä. OPC UA -yhteyskäytäntö on yleisessä käytössä automaationohjelmistoissa, sillä se on OPC Foundation -yhdistyksen ylläpitämä standardi. Tämän vuoksi OPC UA -yhteyskäytäntöä löytyy jo valmiina paljon ajureita ja sen liittäminen uusiin järjestelmiin myös tulevaisuudessa onnistuu vaivatta.

5 Tiedonsiirto

5.1 MQTT

MQTT (MQ Telemetry Transport) on kevyt ja yksinkertainen tiedonsiirtoyhteyskäytäntö, joka perustuu tilaus- ja julkaisutyyppiseen (Publish and subscribe) keskusteluun. MQTT-yhteyskäytännön keksivät Andy Stanford-Clark IBM-yrityksestä ja Arlen Nipper Arcom-yrityksestä (nykyään Eurotech) vuonna 1999. OASIS-yhdistys on standardoinut MQTT-yhteyskäytännön vuonna 2014. OASIS-yhdistys muodostuu yli 6000 jäsenestä ja 600 organisaatiosta, jotka koostuvat eri teknologia-alojen ammattilaisista, ja jotka pyrkivät kehittämään muun muassa teollisen internetin ja pilvipalveluiden standardeja. [9; 10.]



Kuva 5. MQTT-tiedonsiirto muodostuu lähettäjistä, välittäjästä ja tilaajista.

MQTT-yhteyskäytäntö toimii TCP/IP-yhteyskäytännön päällä, ja se on suunniteltu matalille siirtonopeuksille ja pieni resurssisille laitteille sopivaksi. TCP/IP on internetin standardoitu yhteyskäytäntö, jossa annettaan tietoverkon laitteille oma IP-osoite, jonka avulla yhteyskäytäntö luo yhteydet eri laitteiden välille ja välittää IP-paketit niiden välillä. MQTT-yhteyskäytäntö käyttää TCP/IP-yhteyteen porttia 1883, joka on varattu sille TCP/IP-yhteyskäytännön UPD-lähetystasolla, joka vastaa pienikokoisten viestien lähettämisestä [9]. MQTT vaatii toimiakseen MQTT-välittäjän (broker), joka ohjaa tiedonsiirron kulkua. Tilaajat ottavat yhteyttä välittäjään, johon luodaan aiheita (topic). Tilaajat tilaavat aiheita, joihin lähettäjät voivat julkaista tietoa. Kuvassa 5 on havainnollistettu MQTT-viestinnän hierarkiaa. [9; 11.]

MQTT-yhteyskäytäntö lähettää tietonsa binäärisessä muodossa, jossa ensin on pakollinen kiinteä ylätunniste (fixed header), seuraavaksi joissakin paketeissa on muuttuva ylätunniste (variable header) ja lopuksi voi olla tietosisältö (payload). Kiinteällä ylätunnisteella ohjataan, millainen viesti on kyseessä, esimerkiksi onko kyseessä palvelimen yhdistäminen asiakasohjelmalle vai tiedonjulkaisu (publish) ja kuinka pitkä paketti on kyseessä. Muuttuvalla ylätunnisteella voidaan antaa lähetettävälle paketille lisätunniste, jota jotkin lähetystyypit tarvitsevat. Tiedon loppuosassa sijaitseva tietosisältö voi sisältää käyttäjätunnisteen, aiheen, viestin, käyttäjän ja salasanan lähetettävästä viestistä riippuen. [12; 13.]

5.1.1 Mosquitto

Työssä käytettiin Eclipse Mosquitto -nimistä MQTT-välittäjäohjelmaa, jonka avulla voidaan siirtää mittaustietoa Raspberry Pi -tietokoneen ja Windows-tietokoneella pyörivän KepwareServerEx-palvelimen välillä. Mosquitto on avoimen lähdekoodin ohjelmisto, joten se sopii hyvin edullisiin itse tehtyihin ratkaisuihin. Mosquitto käyttää hyväkseen version 3.1 MQTT-yhteyskäytäntöä, joka on uusin tämän tyypin standardoitu yhteyskäytäntö. [14.]

Mosquitto toimii monilla alustoilla kuten Windows- ja Linux-käyttöjärjestelmillä. Mosquitto Broker voidaan asentaa tietokoneelle, jonka jälkeen sitä voidaan käyttää viestien välitykseen. Mosquitto-ohjelmalla on oma asetustiedostonsa, josta voidaan muuttaa ohjelman asetuksia. Asetuksiin sisältyy muun muassa porttien säätäminen, käyttäjätietojen muutokset ja salausmenetelmien asettaminen. [15.]

5.1.2 Paho-MQTT

Eclipse Paho MQTT on Python-kieleen pohjautuva kirjasto MQTT-tilaukseen, jonka avulla voidaan Python-ohjelmakoodeihin rakentaa MQTT-julkaisija. MQTT-viestintää varten ohjelmaan täytyy kirjoittaa asiakasohjelmayhteys välittäjällä, ja yksinkertaisinta yhteyttä varten riittää pelkästään välittäjän IP-osoite Mosquitto-välittäjän oletusasetuksilla. Ohjelmassa tulee olla myös tietoverkkosilmukka, jonka avulla yhteys ylläpidetään. Kun yhteys välittäjään on luotu, voidaan ohjelmaan kirjoittaa julkaisukomento (publish), johon voidaan kirjoittaa merkkijono lähetettäväksi ja aihe (topic), johon merkkijono lähetetään. Näiden avulla viesti siirtyy Paho-MQTT-ohjelmasta välittäjälle. [16.]

5.1.3 KepwareServerEx IoT Gateway

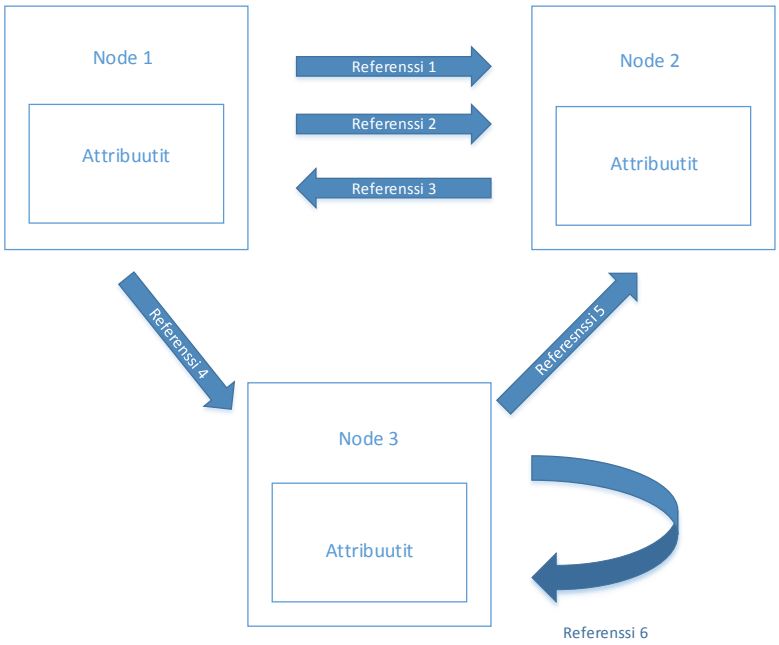
Työssä käytettyyn tietokantaan tiedon siirtämistä varten MQTT-yhteyskäytännöllä tarvitaan KepwareServerEx IoT Gateway -lisäosaa. Yrityksen käytössä oleva historiatietokanta ei tue muuta kuin OPC DA -yhteyskäytännön mukaisia viestejä, joten tarvitaan ohjelma, joka toimii siltana MQTT-viestinnän ja tietokannan välillä. KepwareServerEx-palvelin toimii välittäjänä MQTT- ja OPC DA -tiedonsiirron välillä, jossa MQTT-viestit yhdistyvät KepwareServerEx-ohjelmiston IoT Gateway -lisäosaan. IoT Gateway voidaan asettaa toimimaan MQTT-asiakasohjelmana, joka tilaa Raspberry Pi -tietokoneen lähettämää aihetta. Aiheeseen julkaistaan IoT gateway -ohjelmalle sopivaa tietoa, jonka tulee olla muodossa [{"id": "Channel1.Device1.Tag1","v": 42}], jossa luku 42 on haluttu mittausarvo ja teksti ”Channel1.Device1.Tag1” kertoo ohjelmalle muuttujan, johon tieto halutaan lukea. [17.]

KepwareServerEx 6 -ohjelmassa MQTT-asiakasohjelma lisääminen onnistuu käyttäen ohjelman asetusikkunaa (configuration window). Asetusikkunassa voidaan lisätä IoT Gateway-lisäosaan uusi MQTT-asiakasohjelma, jolle voidaan asettaa muun muassa suojaus- lukunopeus- ja muuttuja-asetukset. Asiakasohjelma vaatii toimiakseen MQTT-välittäjäohjelman, jonka osoite tulee asettaa muodossa tcp://localhost:1883, jossa localhost-kohdan tilalle voidaan asettaa välittäjäohjelmaa pyörittävän laitteen IP-osoite. Asiakasohjelmalle voidaan myös asettaa halutessa turva-asetukset, joihin kuuluvat tunnus, käyttäjä ja salasana. Nämä asetukset on asetettava samalla tavalla myös tietoa lähettävän MQTT-laitteen ohjelmaan, jotta yhteys laitteiden välillä voidaan muodostaa. KepwareServerEx-ohjelmiston IoT Gateway -lisäosaan pitää vielä asettaa muuttujat, joihin tiedot luetaan. Muuttujille annetaan nimi, tietotyyppi ja lukunopeus, jolloin ohjelma on valmis lukemaan MQTT-viestejä näihin muuttujiin.

5.2 OPC Unified Architecture

OPC UA on OPC Foundation -yhdistyksen vuonna 2008 luoma yhteyskäytäntö automaation tiedonsiirtoon, mikä toimii TCP/IP-yhteyskäytännön päällä. OPC UA on kehittyneempi versio vanhemmasta OPC DA -yhteyskäytännöstä, joka toimii pelkästään Windows-ympäristössä COM-tekniikkaan perustuen. OPC UA -tekniikalla pystytään helpommin tunneloimaan automaation tietoverkoissa ja kyetään myös salattuun tiedonsiirtoon. [18]

OPC UA -yhteyskäytännöllä voidaan siirtää automaation tietoliikennettä suojatusti käyttäen sessio- ja tilaustekniikoita. Tiedonsiirto perustuu asiakasohjelma- ja palvelinryhmiin, joissa luodaan sessio asiakasohjelman ja palvelimen välillä. Subscription (tilaus) -tekniikalla voidaan luoda muuttujaan tilaus, jolloin asiakasohjelma pystyy lukemaan tätä muuttujaa. Subscription-tekniikan etuna normaaliin luku- ja kirjoitus-tekniikkaan verrattuna on sen kyky lukea tietoa vain silloin, kun tiedossa tapahtuu muutos. Täten tilaaja saa haluamansa arvon reaaliaikaisesti, kuitenkin siten, ettei turhaa liikennettä synny lukiessa muuttumatonta arvoa.



Kuva 6. OPC UA muodostuu yhtymäkohdista ja referensseistä, joilla on omat attribuuttinsa.

OPC UA -yhteyskäytäntö perustuu yhtymäkohtiin (node) ja referensseihin (kuva 6) [19, s.22]. Yhtymäkohtien avulla kuvataan järjestelmän eri tapahtumat ja tiedot kuten instanssit ja tyypit. Yhtymäkohdat voivat olla erityyppisiä riippuen niiden käyttökohteista, ja niillä on omat attribuuttinsa, joiden avulla niiden toimintaa kuvataan. Yhtymäkohdat yhdistyvät toisiinsa käyttäen referenssejä, jotka kuvaavat yhtymäkohtien suhdetta toisiinsa. Referenssien avulla muodostetaan hierarkia yhtymäkohtien välillä ja kerrotaan, ovatko suhteet näiden välillä yhden suuntaisia, vanhempi- tai lapsisuhteita vai ovatko ne molempiin suuntiin kulkevia sisarsuhteita. Vanhempisuhteen yhtymäkohdat omistavat sen lapset, jolloin lapsi perii myös vanhemman ominaisuudet. Taulukossa 1 on kuvattu OPC UA -standardin kaikkien yhtymäkohtien yleiset attribuutit, mutta eri yhtymäkohtaluokilla on myös omat attribuuttinsa. Referensseillä on myös nämä samat yleiset

attribuutit, joiden lisäksi niillä on attribuutit, jotka kertovat, onko kyseessä pelkästään organisoiva referenssi, onko referenssi samanlainen kumpaankin suuntaan ja miten referenssi toimii käänteisen suuntaan. Eri yhtymäkohdat erotetaan toistaan nodeID-tunnisteen avulla. NodeID-tunnisteessa kuvataan, mihin nimiavaruuteen yhtymäkohta kuuluu ja mikä on sen tunniste tässä nimiavaruudessa. Nimiavaruus on jonkin tahon ylläpitämä tunnistemäärittely, joka pitää huolta, että tunnisteet pysyvät samanlaisina [19, s. 68–69]. Tässä työssä käytettiin OPC UA -palvelimella paikallista nimiavaruutta, muuttujien vähäisen määrän vuoksi. [19, s. 22–29.]

Taulukko 1. OPC UA koostuu erilaisista yhtymäkohdista, joilla on ainakin taulukon kuvaamat pakolliset attribuutit [19, s. 23].

Attribuutti	Tietotyyppi	Kuvaus
NodeId	NodeId	Attribuutti todentaa yhtymäkohdan OPC UA -palvelimella uniikilla tunnuksella ja sitä käytetään yhtymäkohdan käsittelyyn palvelimella.
NodeClass	NodeClass	Attribuutti kuvaa yhtymäkohdan luokan luokiteltujen tyyppien (enumeraation) avulla. Näitä voivat olla muun muassa objektit ja metodit.
BrowseName	Qualified-Name	Attribuutin avulla tunnistetaan yhtymäkohta OPC UA -palvelimen sisältöä selatessa.
DisplayName	Locali-zedText	Attribuutti sisältää yhtymäkohdan nimen, joka näytetään käyttäjälle käyttöliittymässä.
Description	Locali-zedText	Attribuutti sisältää vapaaehtoisen kuvauksen luettavassa tekstimuodossa yhtymäkohdasta.
WriteMask	UInt32	Attribuutti on vapaaehtoinen tieto, jolla voidaan merkitä yhtymäkohtaan mitä sen attribuutteja OPC UA -asiakasohjelma voi muokata.
UserWrite-Mask	UInt32	Attribuutti on vapaaehtoinen tieto, joka kertoo mitä yhtymäkohdan attribuutteja tämän hetkinen palvelimelle kirjautunut käyttäjä saa muokata.

OPC UA -yhteyskäytännön tärkeimpiä yhtymäkohtaluokkia ovat objektit, muuttujat ja metodit. Objekteilla on omat muuttujat ja metodit, joiden avulla objekti pystyy esimerkiksi lukemaan ja kirjoittamaan tietoa. Objekti ei itsessään sisällä tietoa, vaan sitä käytetään osoiteavaruuden strukturointiin. Muuttuja edustaa tiettyä arvoa, jolle on asetettu tietotyyppi, joka kertoo millaista tietoa muuttajaan voidaan tallentaa. Muuttujaluokan yhtymäkohdassa esiintyy

- itse arvo
- tietotyyppi
- taulukkomuotoisuuden asetukset
- taulukon koko
- luku- ja kirjoitusoikeudet
- käyttäjäoikeudet muutoksiin
- minimiaika arvojen muutoksien havainnointiin
- muuttujan historia-arvojen tallennusasetukset [19, s. 32].

Metodien avulla asiakasohjelma pystyy kutsumaan palvelimen tietoja käyttäen parametreja, jotka annetaan metodille. Näiden parametrien avulla metodi palauttaa sen sisällön perusteella asiakasohjelmalle uudet tiedot kuten luetun arvon. Metodiluokan yhtymäkohdissa attribuutteina ovat versionumero, millainen kirjainjärjestelmä on käytössä, enumeraatiot ja millaiset argumentit sopivat sisääntuloille ja ulostuloille. Enumeraatio kuvaa mahdollisia listoja muuttaen numeroarvot tekstiksi, jolloin esimerkiksi numero yksi voi vastata sanaa ”arvo”. [19, s. 114.]

OPC UA -viestintää on myös mahdollista salata käyttäen OPC UA -yhteyskäytännölle määritettyjä salausmenetelmiä. Salaus alkaa OPC UA -palvelimen sertifikaatista, joka on palvelimelle uniikki cert-päätteinen tiedosto. Tämä tiedosto generoidaan palvelimelle, jonka jälkeen se voidaan siirtää asiakasohjelman saataville. Asiakasohjelmaan asetetaan tiedostopolku, josta se pystyy lukemaan tämän sertifikaatin. Kun sertifikaatti on hyväksytty, asiakasohjelma pyytää salattua kanavaa liikennettä varten. Palvelin vaatii kanavaa varten turvallisuustyypin, joka voi olla ”None”, ”Sign” tai ”SignAndEncrypt”. None-tyyppi tarkoittaa sitä, ettei viestiä salata ollenkaan. Sign-tyypillä käytetään salaukseen yksityisavainta, joka on pitkä merkkisarja generoituna OPC UA -asiakasohjelmalle. Jos käytössä on SignAndEncrypt-salaustyyppi, salataan viesti vielä

käyttäen palvelimen julkista avainta, joka kryptaa viestin, jolloin sitä on lähes mahdotonta lukea ilman tätä avainta. Vaikka joku pystyisi sieppaamaan lähetetyn viestin, hänelle se näyttäisi täysin lukukelvottomalta merkkisarjalta. [19, s. 213.]

5.2.1 Python FreeOPCUA

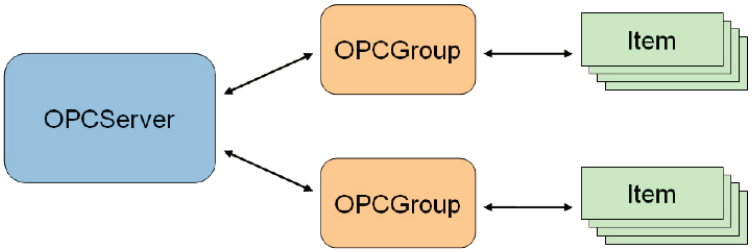
Python FreeOPCUA on Python-ohjelmointikielinen avoimen lähdekoodin OPC UA -palvelin- ja asiakasohjelmakirjasto. Kirjastosta löytyvät tarvittavat metodit yhteyksien luomiseen ja tiedon välitykseen OPC UA -yhteyskäytäntöä käyttäen. Yksinkertaisimmillaan kirjastoa voidaan käyttää OPC UA -palvelimen luomiseen siten, että palvelimen Python-ohjelmaan alustetaan sen loppupiste, objektit ja muuttujat. Palvelimen loppupiste on muotoa opcua.tcp://127.1.1.1:4840/freeopcua/server, jossa palvelimen IP-osoite voidaan asettaa 127.1.1.1 osoitteen tilalle ja 4840 on vakioportti OPC UA -palvelimelle. Loppuosa ”freeopcua/server” on palvelimen nimi sen tunnistamista varten, ja tätä voidaan muokata vapaasti, kunhan sama asetetaan asiakasohjelman asetuksiin. Tämän loppupisteen avulla asiakasohjelmat tunnistavat oikean palvelimen ja pyrkivät ottamaan yhteyttä tähän palvelimeen. Kirjaston muuttujat löytyvät oletusnimiavaruudesta numero kaksi, ja niille voidaan määrittää haluttu numerotunniste palvelimen alustuksessa. [20.]

5.2.2 OPC Data Access

OPC DA on klassisen OPC-yhteyskäytännön yksi määrittely, jonka avulla pystytään siirtämään tietoa eri automaatiojärjestelmien välillä. Klassista OPC:ta ylläpitää OPC-foundation-säätiö, joka on standardisoinut automaation OPC-viestinnän. Tämän vuoksi OPC on levinnyt erittäin laajalle ja löytyy melkein kaikista automaationjärjestelmistä vaihtoehtoisena yhteyskäytäntönä. Klassinen OPC luotiin vuonna 1996 Windows-käyttöjärjestelmää silmälläpitäen, joten se käyttää Windowsin COM-viestintämenetelmiä tiedonsiirtoon [19, s. 1]. Tämän vuoksi klassinen OPC toimii vain Windows-järjestelmissä ja sitä ei pystytä helposti tunneloimaan automaation tietoverkoissa [19, s. 4]. Tästä syystä on siirrytty uudempiin yhteyskäytäntöihin ja OPC Foundation -säätiö loi myös oman uuden version OPC:sta nimeltä OPC Unified Architechture. OPC-standardissa on mukana yli 450 automaatiojärjestelmien tarjoajaa jäsenenä, ja OPC-tuotteita on olemassa yli 15 000 kappaletta. Tämä yhteyskäytäntö on samalla tavalla levinnyt uusissa järjestelmissä laajalle OPC UA -yhteyskäytännön avulla ja tulee

todennäköisesti tulevaisuudessakin olemaan mukana useiden toimittajien järjestelmissä. [19, s. 2.]

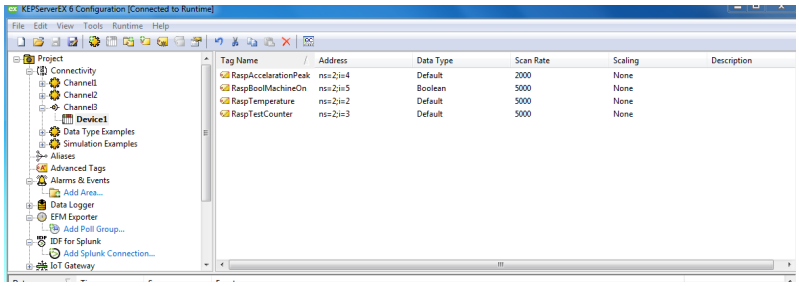
Klassinen OPC perustuu palvelin-asiakasohjelmayhteyteen, jonka avulla järjestelmä pystyy kirjoittamaan ja lukemaan tietoa. Yhteys luodaan käyttäen OPC-ryhmiä (kuva 7), joiden avulla voidaan selata eri metodeita käyttäen palvelimen osoiteavaruutta, jossa palvelimen muuttujat sijaitsevat. Muuttujilla on ominaisuuksina sen arvo, tietotyyppi ja käyttöoikeudet, joita voidaan OPC-asiakasohjelmalla lukea. Näin asiakasohjelma voi lukea haluttujen muuttujien arvoa reaaliaikaisesti.



Kuva 7. OPC DA -yhteyskäytännössä palvelin jakautuu ryhmiin, joilla on omat muuttujat.

5.3 KepwareServerEx

KepwareServerEx on ohjelmisto, jolla luodaan automaatio-ohjelmistojen välisiä viestintäyhteyksiä. KepwareServerEx-ohjelmasta löytyy monipuolisesti eri automaatiolaitteiden ja -järjestelmien ajureita, joiden avulla yhteydet luodaan. Ohjelma sisältää myös OPC UA ja OPC DA -palvelimet ja asiakasohjelmat sekä MQTT-viestintää varten IoT Gatewayn, joita tässäkin työssä käytetään. Ohjelmaa voidaan käyttää siltana erilaisten järjestelmien välillä, jolloin etuna on sen monipuolinen määrä yhteyskäytäntöjä. [21.]



Kuva 8. KepwareServerEX 6 -ohjelmiston asetusnäkymä

KepwareServerEx-ohjelman asetuksia voidaan muokata asetusikkunassa (kuva 8). Asetusikkunassa voidaan asettaa ohjelmalle projekti, johon voidaan tallentaa halutut asetukset. Yhteyksiä varten on olemassa connectivity-kohta, jolle voidaan asettaa eri kanavia. Kanaville asetetaan halutut ajurit, joiden avulla viestinnän lukeminen toteutetaan. Kanavalle lisätään vielä laite (device), jonka avulla voidaan tarkentaa käytetyn viestintätavan asetuksia tietylle laitteelle tai yhteydelle.

5.4 Historiatietokanta

Tiedon tallennukseen työssä käytettiin historiatietokantaa. Tieto siirrettään tietokantaan käyttäen sen omia keruuajureita, jotka lukevat tiedonsiirtopalvelimelta halutut muuttujat. Kyseinen tietokanta pystyy keskustelemaan vain käyttäen OPC DA -yhteyskäytäntöä, joten tämä vaatii OPC DA -palvelimen toimiakseen. Historiatietokantaan voidaan asettaa, kuinka usein tietokanta lukee OPC DA -palvelinta ja mitä muuttujia se lukee, mikä rajoittaa tietoliikenteen määrää vähentämällä lukupyyntöjä tiedonsiirtoväylässä. Tietokannassa tietoa voidaan muokata haluttuun muotoon käyttäen erillisiä muokkaustyökaluja, jotka pystyvät suodattamaan turhan tiedon pois ja luomaan mittatiedosta graafisia kuvaajia.

Historiatietokantoja käytetään tiedontallennukseen. Kun tietoa halutaan tallentaa suuria määriä, täytyy tällaisissa sovelluksissa käyttää tietokantaohjelmistoja tiedonkeräämiseen. Tietokantaohjelmistossa tieto tallennetaan taulukkoon, jossa jokaisella rivillä on oma tunnistekoodinsa. Tämän koodin avulla tieto voidaan tallentaa tietyille riveille, josta sitä voidaan tietokantahakukielillä lukea hakemalla käyttäen tunnistekoodia. Historiatietokannassa tieto tallennetaan tunnisteiden mukaan eri muuttujille, johon tallennetaan tunnisteen nimi, arvo, tiedon laatu ja aikaleima. [22, s. 74–76.]

Historiatietokannan hyötynä on sen yksinkertaisempi tallennusrakenne verrattuna relaatiotietokantoihin. Relaatiotietokannassa tieto tallennetaan tietyille riveille numerotunnisteen avulla, jossa taulu voi liittyä toisiin tauluihin avainten avulla [23]. Avain kertoo, mistä taulusta alkuperäinen tieto on lähtöisin. Historiatietokanta tallentaa tiedon aikaleimalla tunnisteiden mukaan, jolloin tiedon tallentamisessa ei tarvitse miettiä taulujen rakennetta ja liittymistä toisiin tauluihin. Kunnonvalvontamittauksiin tämä sopii hyvin, sillä kunnonvalvonnassa halutaan verrata mittaustiedon kehittymistä ajan kuluessa ja tietoa halutaan kerätä erittäin paljon. [22, s. 7.]

6 Laitteisto

6.1 Raspberry Pi

Raspberry Pi on pieni (85 x 56 mm) ja edullinen mikrokontrolleri, johon voidaan asentaa Linux-pohjainen käyttöliittymä. Raspberry Pi on siis pienoistietokone, jossa keskusmuistia on yhden gigatavun verran ja joka sisältää 1,2 gigahertsin nopeudella toimivan neliydinprosessorin. Raspberry Pi -tietokoneelle on mahdollista asentaa Linux-pohjaisia ohjelmistoja ja käyttää sitä kuin normaalia Linux-tietokonetta. Työssä käytetään Raspberry Pi 3B -tyypin tietokonetta, mutta laitteesta on myös vanhempia versioita. Uusimman version etuna on sen kehittynyt tehokkuus, joka ei kuitenkaan tuo laitteelle juurikaan lisähintaa verrattuna vanhempiin versioihin. [24.]

Raspberry Pi 3B -malli sisältää Ethernet-portin, WLAN-sirun, Bluetooth-vastaanottimen, digitaaliset sisääntuloportit, HDMI-liittimen, neljä USB-liitintä ja muistikortinlukijan. Kuitenkin lisäosilla pystytään antamaan tietokoneelle lisäominaisuuksia kuten Power over Ethernet -yhteys tai analogiset sisääntulot. Muistikortille voidaan asentaa Linux-käyttöjärjestelmä sekä muut halutut ohjelmat. Liittämällä näyttö HDMI-liittimen avulla Raspberry Pi -tietokoneeseen voidaan tietokoneen käyttöliittymää käyttää sen ohjaamiseen, mutta yhteys voidaan myös luoda käyttäen SSH-yhteyskäytäntöä, jolloin erillistä näyttöä ei tarvitse laitteeseen kytkeä. SSH-yhteyskäytännön avulla voidaan luoda salattu yhteys kahden laitteen välille, jotka ovat kytkettynä samaan tietoverkkoon. Raspberry Pi -tietokoneelta pitää ensin kytkeä SSH-ominaisuus päälle, jolloin käyttäjän ja salasanan avulla voidaan luoda yhteys samassa tietoverkossa olevalta tietokoneelta käyttäen SSH-yhteyteen kykenevää ohjelmistoa kuten Putty-ohjelmistoa. [25; 26.]

Raspberry Pi (kuva 9) kykenee kytkeytymään sisäisiin tai ulkoisiin tietoverkkoihin sen WLAN- ja Ethernet-kytkentöjen avulla. WLAN-yhteydellä tarkoitetaan langatonta yhteyden luontia tietoverkossa ja Ethernet-yhteydellä tarkoitetaan Ethernet-kaapelilla yhdistettyjä fyysisiä tietoverkkoja. WLAN-yhteydet tarvitsevat toimiakseen erillisen WLAN-tukiaseman, johon Raspberry Pi voi ottaa yhteyttä. Tukiasemalta yhteys jatketaan tietoverkon muihin osiin Ethernet-kaapelia pitkin. Raspberry Pi -tietokoneelle on mahdollista luoda dynaaminen tai kiinteä IP-osoite riippuen laitteen käyttökohteesta. Kiinteällä osoitteella voidaan varmistaa, että laitteen osoite pysyy vakiona, jolloin on helpompaa löytää se tietoverkosta. Dynaamisella osoitteella DHCP-palvelin jakaa automaattisesti sen tietoverkon osalle osoitettuja IP-osoitteita, jolloin IP-osoitteet saattavat muuttua, mutta uusien laitteiden liittäminen verkkoon on nopeampaa.

Kuva 9. Raspberry Pi -mikrokontrollerilevy [27].

Raspberry Pi -tietokoneessa on digitaalisia sisääntuloja, joiden avulla voidaan mitata esimerkiksi lämpötilaa tai kiihtyvyyttä. Kuitenkaan analoginen lämpötilan mittaus ei onnistu suoraan ilman tiedon muunnosta digitaaliseksi. Viestintään Raspberry Pi käyttää I2C- ja SPI-yhteyskäytäntöjä. Analoginen mittaustieto käännetään tietokoneelle sopiviksi digitaalisiksi arvoiksi AD-muuntimen avulla. AD-muunnin on analogisesta digitaaliseen muuntava piirikortti, joka muuntaa analogisen jännitearvon digitaaliseen tavumuotoon nolliksi ja ykkösiksi. AD-muuntimella on bittiresoluutio, joka jakaa jännitearvon resoluution määräämiin osiin niin, että yksi bitti vastaa yhtä askelta jännitteen kasvussa. Tällöin maksimijännite muunnetaan niin, että kaikki bitit ovat pystyssä ja nollajännitteellä kaikki bitit ovat nollassa. Analoginen anturi saa jännitteensä Raspberry Pi -tietokoneen jännitelähdöistä. Näiden sisääntulojen heikkoutena on niiden matala jännitetaso, joten laitteeseen tarvitsee hankkia antureita, jotka kykenevät toimimaan 3,3 tai 5 voltin jännitteellä. Lisälevyillä on kuitenkin mahdollista toteuttaa mittauksia, jotka käyttävät esimerkiksi 4–20 milliampeerin virtaviestejä.

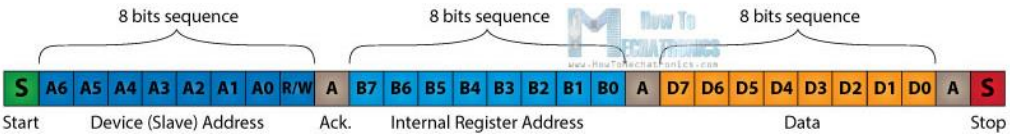
Raspberry Pi -tietokoneella on useampi tapa lukea sen digitaalisia sisääntuloja kuten I2C- ja SPI-tiedonsiirtoyhteyskäytännöt. Nämä yhteyskäytännöt löytyvät valmiiksi asennettuina Raspberry Pi –tietokoneella, ja ne voidaan kytkeä päälle tietokoneen asetuksista. Nämä yhteyskäytännöt perustuvat useamman digitaalisen tulon yhtäaikaiseen lukemiseen. Näiden digitaalisten tulojen avulla yhteyskäytännöt pystyvät lukemaan yhtä tai useampaa analogista tietoa käyttäen näille käytännöille luotuja ohjelmointikirjastoja ja ohjelmia. [28.]

I2C on yksi Raspberry Pi -tietokoneessa toimivista yhteyskäytännöistä tiedonsiirtoon antureiden kanssa. I2C:n avulla voidaan kahden digitaalisen sisääntulon avulla siirtää analogista tietoa. Heikkoutena yhteyskäytännössä on sen lukuhitaus verrattuna SPI-yhteyskäytäntöön. Kuitenkin yhteyskäytäntö on toimintavarmempi pidemmillä johdotuksilla ja ajureiden saatavuus on helpompaa yhteyskäytännön standardoinnin vuoksi. [29.]

I2C:n avulla voidaan yhdistää samoilla johtimilla useita eri laitteita, joista toiset ovat orjia ja toiset ovat isäntiä. Isännät ohjaavat liikennettä ja orjat tottelevat isäntien käskyjä lähettämällä viestinsä, kun isäntä kutsuu orjaa sen osoitteella. Viestit lähetetään antureissa olevien SDA- ja SCL-ulostulojen avulla. SCL toimii signaalin kellona, jolla ajoitetaan tiedon siirtäminen. Itse tieto saadaan SDA-ulostulon korkeasta ja matalasta jännitteestä vastaten binäärisen järjestelmän ykköstä ja nollaa. [29.]

Lähetettävä viesti (kuva 10) sisältää

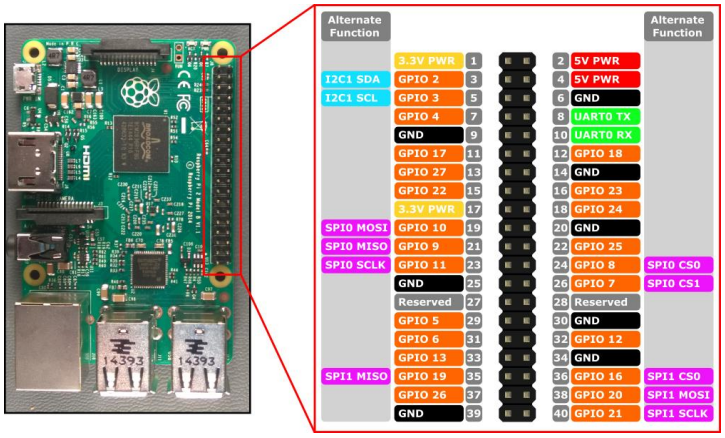
- aloitussignaalin
- 8-bittisen orjan osoitteen
- 8-bittisen orjan sisäisen rekisterin tiedot
- 8-bittisen lähetettävän tiedon
- kuittauksen viestin saannista jokaisen 8-bittisen osan jälkeen
- lopetussignaalin.



Kuva 10. I2C-yhteskäytänntö lähettää kolme 8 bitin sarjaa viesteissään [29].

Aloitussignaali kertoo, milloin viesti alkaa, ja lopetussignaali, milloin viesti loppuu. Jokaisella orjalla on oma osoitteensa, jonka avulla isäntä kutsuu sitä. Orjan sisäinen rekisteri sisältää tietoja orjan tarvitsemista tiedoista, kuten 3-suuntakiihtyvyysanturin eri suuntien mittausten osoitteet. Viimeinen 8 bitin sarjan mukana lähetetään itse tieto, jota esimerkiksi on anturin mittaama kiihtyvyys. [29.]

SPI-yhteyskäytäntö on toinen Raspberry Pi -tietokoneen yhteyskäytännöistä antureiden tietojen lukemiseen. Myös tämän yhteyskäytännön avulla voidaan siirtää tietoa analogisesta anturista Raspberry Pi -tietokoneen digitaalisiin sisääntuloihin. SPI vaatii kolme tai neljä digitaalista sisääntuloa, joiden avulla voidaan ohjelmallisesti yhdistää digitaalisten tulojen arvot yhdeksi luvuksi. SPI-yhteyskäytännölle on vaikeampaa löytää ajureita kuin I2C-yhteyskäytännölle, koska SPI-yhteyskäytäntöä ei ole standardoitu. SPI on myös paljon herkempi häiriöille. Häiriöitä signaaliin voivat aiheuttaa esimerkiksi liian pitkät johdot tai laitteiston läheisyydessä sijaitsevat voimakkaat sähkömagneettiset kentät. Hyvänä puolena SPI-yhteyskäytännössä on sen lukunopeus, joka on huomattavasti nopeampi kuin I2C-yhteyskäytännöllä. Työssä testatulla kiihtyvyysanturilla pystyttiin SPI-yhteyskäytännöllä mittaamaan kiihtyvyyttä 3200 kertaa sekunnissa, joka on tämän anturin maksimi mittausnopeus. Toisaalta I2C kykeni mittaamaan kiihtyvyyttä maksimissaan 1600 kertaa sekunnissa. [28; 30.]



Kuva 11. Raspberry Pi -tietokoneen sisääntulojen järjestys.

SPI käyttää I2C:n tapaan isäntä-orjayhteyttä, mutta se kykenee vain yhteen isäntälaitteeseen kerralla. SPI-yhteyskäytännöllä on kuitenkin mahdollista liittää useampia orjalaitteita yhteen isäntään, joita isäntälaite ohjaa orjaosoitteiden avulla. Tässä työssä Raspberry Pi -tietokoneen SPI controller -yhteyskäyntäntö toimii käyttämällä yhteensä neljää ulostuloa, jotka ovat Raspberry Pi -tietokoneessa MOSI-, MISO-, CS- ja SCLK-

ulostulot (kuvassa 11). CS-tulo (chip select) ohjaa, milloin viestintä aloitetaan ja lopetetaan niin, että signaalin laskiessa alas viestintä aloitetaan ja se lopetetaan, kun tulon signaali nousee takaisin ylös. CS-tuloja on Raspberry Pi -tietokoneella oletusarvoisesti kaksi kappaletta, joiden avulla valitaan, kumman orjan kanssa isäntä keskustelee. MOSI-tulon avulla lähetään isännän ohjausviestit orjalle ja MISO-tulon avulla saadaan orjan (eli anturin) lähettämä tieto isännälle. SCLK-tulo toimii viestinnän kellona, jonka signaali sykkii tietyllä taajuudella päälle ja pois. Tämä signaali kertoo, missä vaiheessa viestiä ollaan menossa laskemalla signaalin huippujen määrää. Lähetetty tieto sisältää ensin orjan osoitteen, jonka jälkeen lähetetään tieto tavuina. [30; 31.]

6.2 Arduino

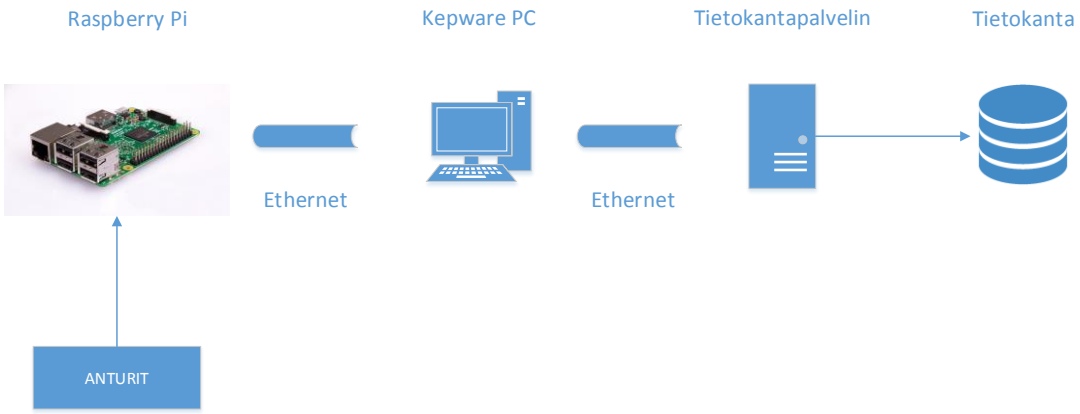
Arduino (kuva 12) on mikrokontrolleri, joka vastaa suurilta osin Raspberry Pi -tietokonetta. Arduino-tietokoneesta on monenlaisia erilaisia versioita, jotka eroavat toisistaan ominaisuuksien osalta. Työssä testattiin Arduino Yun -levyä, sen Ethernet-yhteyden vuoksi. Arduino Yun sisältää Ethernet-yhteyden lisäksi analogiset ja digitaaliset sisääntulot, USB-liitännän laitteen ohjelmointia varten, muistikorttipaikan ja WLAN-yhteyden. Syöttöjänniteensä Arduino saa 5 voltin mini-USB- tai USB-liitännöistä. Arduino-tietokoneen etuna ovat sen valmiiksi asennetut analogiset tulot, joihin voidaan suoraan liittää analogista signaalia lähettäviä antureita. Heikkoutena Arduino-tietokoneella on sen vähäisempi muokattavuus, sillä Arduino tarvitsee oman ohjelmointiympäristönsä sen ohjelmointiin. Työn tapauksessa ei huomattu tarvittavaa hyötyä suorista analogisista tuloista, jotka eivät ilman lisäosia toimi 4–20 mA:n virtaviestin vastaanottamisessa. Raspberry Pi -tietokonetta on mahdollista muokata nyt ja tulevaisuudessa lisälevyjen ja sen joustavan Linux-ympäristön avulla helpommin kuin Arduino-tietokonetta. [32.]

Kuva 12. Työssä testattiin Arduino Yun -mikrokontrolleria [33].

7 Laitteen toteutus

7.1 Testausympäristö

Kunnonvalvontalaitteiston toimintaa testattiin käyttäen Raspberry Pi -tietokonetta mittausalustana, johon anturit kiinnitettiin. Anturien analogiset viestit muutettiin luvuiksi lukemalla anturitieto AD-muuntimelta I2C- ja SPI-yhteyskäyntäntöjä ja muuttamalla saatu bittitieto liukuluvuksi Python-ohjelmassa tai C-ohjelmassa. Samaan Python-ohjelmaan ohjelmoitiin OPC UA -palvelin, johon luotiin objekti ja kolme eri muuttujaa. Näihin muuttujiin asetettiin antureilta mitatut arvot kuten lämpötila, kiihtyvyys ja käyntitieto. OPC UA -palvelinta luettiin käyttäen KepwareServerEx-ohjelmiston OPC UA client -ajuria. KepwareServerEx-ohjelmisto pyöri erillisellä Windows-tietokoneella. Tämä tietokone yhdistettiin palvelimeen, jossa historiatietokanta pyöri. Tiedonsiirto tietokantapalvelimelle suoritettiin käyttäen tietokannan OPC DA collector -lukija-ajureita, joiden avulla tiedot siirrettiin KepwareServerEx-ohjelmiston OPC DA -palvelimelta tietokantaan. Tietoliikenne suoritettiin käyttäen langallisia Ethernet-yhteyksiä, jotka toivat varmuutta tiedonsiirtoon langattomiin yhteyksiin verrattuna. Kuvassa 13 kuvataan testiympäristön rakenne, jossa anturit kytkeytyvät Raspberry Pi -tietokoneelle, Raspberry Pi kytkettiin Ethernet-kaapelilla KepwareServerEx-ohjelmaa pyörittävällä tietokoneelle, joka kytkettiin tietokantaan toisella Ethernet-kaapellilla.



Kuva 13. Testausympäristössä käytettiin kuvan mukaista rakennetta.

Tiedonsiirtoon KepwareServerEx-ohjelmiston ja mittauslaitteen välillä testattiin myös MQTT-viestintää sekä OPC UA -asiakasohjelman sijoittamista Raspberry Pi -tietokoneelle. Molemmilla tavoilla tieto saatiin siirrettyä KepwareServerEx-ohjelmistolle. Erona huomattiin, että jos OPC UA -asiakasohjelma on sijoitettuna Raspberry Pi -tietokoneella, KepwareServerEx-palvelin ei osaa tulkita tietoa huonolaatuiseksi, kun yhteys katkeaa Raspberry Pi -tietokoneelle. Tällöin mittaus näyttää aina hyvälaatuiselta, vaikka Raspberry ei lähettäisikään mitään. Jos palvelin toteutetaan Rasberry Pi -tietokoneella, merkitsee KepwareServerEx-asiakasohjelma katkenneen yhteyden aikana mittausten arvot huonoiksi, jolloin ne voidaan suodattaa pois tietokannan raportointityökaluissa.

Laitteiston toimintaa voidaan tarkkailla joko tarkastelemalla muuttujien laatua tai luomalla muuttuja, joka muuttaa arvoaan koko ajan. Tietokannassa voidaan tutkia näitä muuttujia ja luoda hälytys, kun muuttujan arvo ei muutu tai mittaustietojen laatu muuttuu huonoksi. Muuttujan laadun ollessa huono voidaan arvioida, että mittalaitteisto ei saa virtaa tai yhteys on katkennut. Ohjelmistot pidetään Raspberry Pi -tietokoneella päällä käyttäen sen crontab-ohjelmaa, jonka avulla voidaan määritellä, milloin ohjelmat käynnistetään. Crontab-ohjelmaan voidaan asettaa, että halutut ohjelmat käynnistyvät Raspberry Pi -tietokoneen käynnistyessä. Raspberry Pi -tietokone käynnistyy automaattisesti aina kun siihen kytketään virta, jolloin ohjelmistot voidaan käynnistää uudelleen virran uudelleenkytkennällä. Raspberry Pi -tietokoneen vikoja voidaan tarkastella myös luomalla etäyhteys siihen käyttäen SSH-yhteyteyttä. Tällöin voidaan IP-osoitteen, käyttäjätunnuksen ja salasanan avulla kirjautua Raspberry Pi -tietokoneelle etäyhteydellä ja käyttää Raspberry Pi -tietokoneen Linux-komentoikkunaa toiselta tietokoneelta.

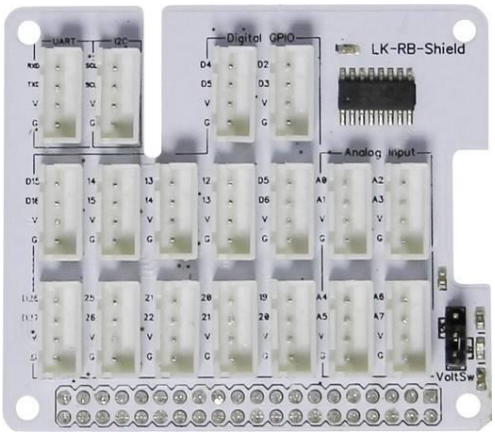
7.2 Ohjelmistot

Työhön valikoitui viisi eri avoimen lähdekoodin Python- ja C-kirjastoa niiden helppokäyttöisyyden, yksinkertaisuuden ja toimivuuden perusteella. Nämä kirjastot olivat Paho-MQTT, FreeOPCUA, ADXL345-python, ADXL345spi ja SPIdev. FreeOPCUA on OPC UA -yhteyksiin rakennettu ilmainen avoimenlähdekoodin Python-ohjelmointikirjasto. Kirjasto käyttää OPC UA -standardin mukaisia metodeita, joiden avulla se pystyy luomaan OPC UA -palvelimia tai asiakasohjelmia. Paho-MQTT-Python-kirjastolla voidaan kirjoittaa ja lukea MQTT-viestejä välittäjältä julkaisu- ja tilausmetodien avulla. ADXL345-python on I2C-yhteyskäytännällä toimiva Python-

ohjelmointikirjasto, jolla voidaan lukea ADXL345-kiihtyvyysanturin analogista jännite-viestiä. Ohjelma muuntaa kahden digitaalisen tulon avulla mitatun arvon kiihtyvyysarvoon. ADXL345spi on C-ohjelmointikielellä luotu ajuri ADXL345-kiihtyvyysanturin lukemiseen SPI-yhteyskäytännöllä. SPIdev taas on Python-kirjasto, jonka avulla pystytään muuntamaan analogiset anturitiedot digitaalisiksi käyttäen SPI-yhteyskäytäntöä. [16; 34; 35; 36.]

7.3 Testattavat anturit

Antureiden testaamista varten laitteistossa oli kytkettynä testattavat anturit, Ethernet-kaapeli ja virtalähde. Raspberry Pi tarvitsee käyttöönsä 5 voltin jännitteen, josta kytkeytyy antureille 3,3 voltin jännite. Anturit kytkeytyvät Raspberry Pi -tietokoneelle joko suoraan sen digitaalisiin tuloihin tai LinkerKit-lisälevyn (kuva 14) kautta. LinkerKit-lisälevy helpottaa analogisten antureiden asentamista Raspberry Pi -tietokoneelle sen painettavilla liitäntäpisteillä. LinkerKit-lisälevy kykenee muuttamaan anturin analogisen 0–3,3 voltin mittauksen Raspberry Pi -tietokoneen tarvitsemaan digitaaliseen muotoon käyttäen sen analogisesta digitaaliseen kääntävää AD-muunninta. Anturit, joita kunnonvalvontalaitteiston testaamiseen käytettiin, valittiin osoittamaan laitteen kykyä mitata erilaisia arvoja, ja ne eivät ole välttämättömiä lopullisen ratkaisun valmistukseen. Antureiden signaalit pitää kuitenkin pystyä muuttamaan digitaalisiksi, ja ne eivät saisi ylittää Raspberry Pi -tietokoneen sisääntulojen 3,3 voltin maksimijännitettä.



Kuva 14. Linkerkit LK-RB-shield -lisälevy helpottaa valmiiden analogisten antureiden kytkentää Raspberry Pi -tietokoneen digitaalisiin sisääntuloihin [37].

LinkerKit-lisälevyyn on mahdollista kytkeä erilaisia LinkerKit-antureita, joista yksi on LK-temp-lämpötila-anturi. Tämän anturin etuna on yksinkertainen kytkentä suoraan painettavilla liittimillä LinkerKit-lisälevyyn, jolloin kytkentöihin ei välttämättä tarvita ruuvaamista ja juottamista. Toisaalta käytettäessä LinkerKitin omia pikaliittimillä varustettuja johtimia on niiden pituus vakioltaan lyhyt, joten anturin asentaminen mitattavan laitteen läheisyyteen ei ole mahdollista näillä johtimilla. Vaihtoehtoisesti lisälevyyn voidaan käyttää johtimia, jotka mahtuvat lisälevyn liittimiin omilla pikaliittimillä.

Tiedonsiirtoon LK-temp käyttää Raspberry Pi -tietokoneen SPI-yhteyskäytäntöä. Tiedon muuttaminen tapahtuu Raspberry Pi -tietokoneelle ohjelmoidussa ohjelmassa, jossa digitaalinen arvo luetaan ja muutetaan jännitearvoksi. Jotta anturin lähettämä tieto saadaan näyttämään mitattua jännitearvoa, tulee se jakaa 1024:llä, joka on anturin bittiresoluutio. Tämä johtuu LinkerKit-lisälevyn AD-muuntimen 8-bittisestä bittiresoluutiosta, jossa tieto siirrettään 8 bitin osissa. LK-temp-lämpötila-anturi (kuva 15) mittaa lämpötilaa alueelta –50…+280 °C. Anturin mittaama analoginen jännitearvo pitää skaalata käyttäen kaavaa

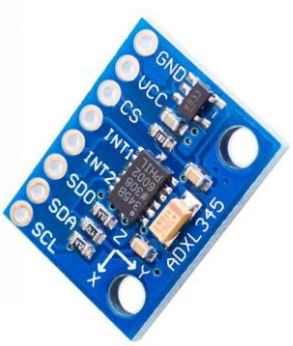
$$\left(\frac{x \times U}{bittiresoluutio} \times 100\right) + t_{min}, \tag{1}$$

jossa x on binääriluku, U on syöttöjännite, t_{min} on minimilämpötila.

AD-muunnin muuttaa anturin mittaaman lämpötilan 8-bittiseksi luvuksi, jolloin luku on välillä 0–1024, joka asetetaan kaavassa 1 muuttujaan ”x”. Raspberry Pi -tietokone syöttää anturille 3,3 voltin jännitteen, jolloin kaavaan voidaan asettaa muuttuja ”U”. Bittiresoluutio määräytyy AD-muuntimen mukaan, joka on kyseisellä muuntimella yhden tavun eli 8 bitin kokoinen ja lukuarvona 1024. Anturin mittaama minimilämpötila on tässä tapauksessa –50 astetta. [31.]

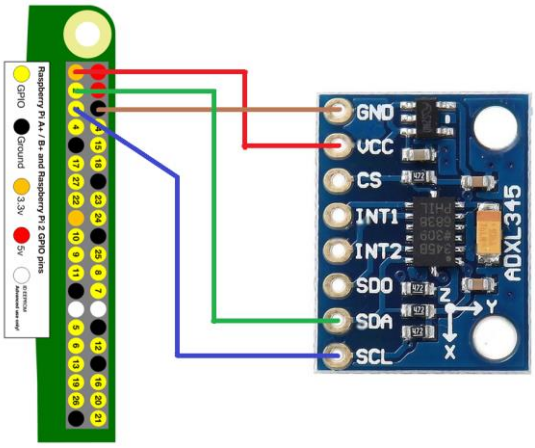
Kuva 15. Työssä testattiin LK-temp-lämpötila-anturia lämpötilan mittaamiseen [38].

Toinen testattava anturi oli ADXL345-kiihtyvyysanturi (kuva 16), joka sisältää 3-akselisen kiihtyvyysanturin. Kiihtyvyysanturin avulla voidaan mitata laitteen värähtelyä mittaamalla kiihtyvyyden huipusta huippuun muutosta. Kiihtyvyysanturi toimii MEMS-tekniikkaa hyväksikäyttäen, laskemalla kiihtyvyyden anturin sisäisten silikoniosien liikkeestä johtuvien resistanssimuutosten avulla. Anturi toimii +/- 2, 4, 8 tai 16 g:n voimien alueella, joista voidaan valita haluttu alue ohjelmallisesti. Mittausresoluutio anturin AD-muuntimella on 13 bittiä, eli se pystyy mittamaan 8192 askeleella kiihtyvyysarvoansa. Tiedonsiirtoon anturi käyttää digitaalisia I2C- tai SPI-yhteyskäytäntöjä. Anturilla pystytään mittaamaan maksimissaan 3200 kertaa sekunnissa, mutta testeissä I2C-yhteyskäytännöllä ei pystytty lukemaan anturia kuin maksimissaan 700 kertaa sekunnissa. SPI-yhteyskäytännön avulla päästään anturin maksimi lukunopeuteen. [30, s. 6.]

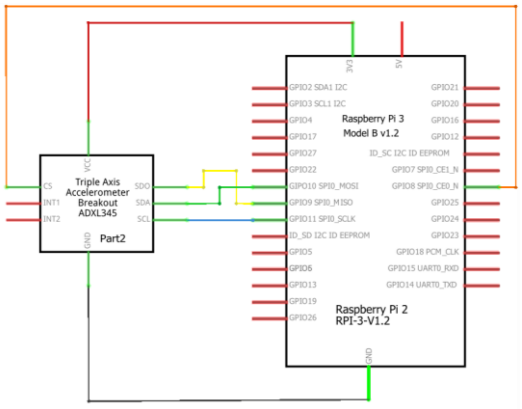


Kuva 16. Tärinän mittausta varten kokeiltiin ADXL345-kiihtyvyysanturia [40].

Raspberry Pi -tietokoneelle ADXL345-kiihtyvyysanturi kytkettiin I2C-yhteyskäytöä varten siten, että anturin GND kytkettiin Raspberry Pi -tietokoneen maahan eli GND-tuloon, VCC kytkettiin Raspberry Pi -tietokoneen 3,3 voltin jänniteulostuloon, SDA kytkettiin Raspberry Pi -tietokoneen sisääntuloon numero 2 ja SCL numeroon 3. Nämä sisääntulot ovat normaalien digitaalisten viestien lisäksi varattu I2C-viestintää varten. Kuvassa 17 esitetään kaaviokuva, kuinka ADXL345 kytketään Raspberry Pi -tietokoneelle I2C-yhteyskäytäntöä käyttäen. SPI-yhteyskäytännöllä ADXL345-anturi kytketään käyttäen Raspberry Pi -tietokoneen sisääntuloja 10, 9, 11 ja 8. Nämä sisääntulot on varattu SPI-yhteyskäytännön MOSI, MISO, SCLK ja CS0 sisääntuloille. ADXL345-levylle nämä sisääntulot kytketään kuvan 18 mukaan. Näiden lisäksi levylle tulee kytkeä 3,3 voltin syöttöjännite ja maadoitus.



Kuva 17. Kytkentäkaavio ADXL345-anturin kytkemisestä I2C-viestintää varten Raspberry Pi -tietokoneelle.



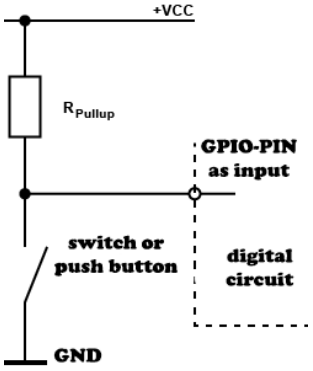
Kuva 18. Kytkentäkaavio ADXL345-anturille Raspberry Pi -tietokoneeseen SPI-yhteyskäytännöllä [35].

Raspberry Pi -tietokoneen digitaalisten GPIO-tulojen avulla pystytään myös mittaamaan esimerkiksi mitattavien laitteiden käyntiaikoja. Raspberry Pi -tietokoneella on 8 vapaata sisään- tai ulostuloa tällaiselle mittaukselle. Sisääntulot on ohjelmoitava GPIO.setup-metodilla, jolla määritetään haluttu kytkentäpiste, onko kyseessä sisään vai ulostulo ja miten halutaan sisääntulon reagoivan piirin katkeamiseen. Ohjelmassa tämä näyttää tältä:

```
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Esimerkkikoodi 1. Raspberry Pi -tietokoneen sisääntulon asettaminen.

Pull_up_down-muuttujalla määritetään ylösvetovastuksen tila eli mitä piirikytkentää ylösvetovastus käyttää. Tässä on mahdollista ohjelmoida sisääntulot pakotetusti tilaan, jossa ne ovat joko ylä- tai alatilassa kun virtapiiri on auki. Sisääntuloihin on kytkettynä vastuksia, jotka ohjaavat sisääntulon näissä tilanteissa 3,3 volttiin tai maapotentiaaliin. Kuvassa 19 esitetään, kuinka ylösvetovastus on kytketty Raspberry Pi -tietokoneelle. Päälle/Pois-kytkintä tehdessä yleisesti käytetään ylätilaan vedettävää vastusta, jolloin vastus pakottaa jännitteen päälle sisääntuloon, kun virtapiiri on auki. Tällöin ei synny ylimääräisiä jännitepiikkejä, koska kytkinpiiri kytketään sisääntuloon ja maahan, jolloin ylimääräiset jännitteet ohjautuvat maapisteeseen. [41.]



Kuva 19. Raspberry Pi -tietokoneen sisääntuloissa on mahdollista käyttää sisäänrakennettua ylösvetovastusta ohjaamaan jännite korkeaksi piirin ollessa auki [41].

Raspberry Pi -tietokoneen sisään tulojen käyttämää 3,3 voltin jännitettä ei aina liitettävästä laitteesta löydy, joten on käytettävä apurelettä tai muuta vastaavaa tasosovitusta ohjaamaan Raspberry Pi -tietokoneen sisääntuloa. Kun mitattavalta laitteelta saadaan signaali, se sulkee releen kytkimen, jolloin Raspberry Pi -tietokoneen maa kytkeytyy haluttuun sisääntuloon. Käyttämällä tällaista erotinta, voidaan Raspberry Pi -tietokoneen omaa jänniteulostuloa käyttää ohjaamaan sen yhtä digitaalista sisääntuloa. Käyntiajan mittaamisen lisäksi käyntitiedon avulla voidaan tutkia, milloin laite on ollut sammutettuna, jolloin voidaan tältä ajalta suodattaa muista mittauksista mittausarvot pois aikajanalta. [42.]

Erilaisilla lisälevyillä on myös mahdollista lisätä vaihtoehtoisia analogisia antureita Raspberry Pi -tietokoneelle. Lisälevyillä on suoraan liitettynä AD-muunnin, joka pystyy muuntamaan analogisen signaalin digitaaliseksi. Raspberry Pi -tietokoneen I2C- ja SPI-yhteyskäytäntöjen avulla voidaan lukea lisälevyiltä saatavat digitaaliset viestit ja ohjelmassa muuntaa ne ymmärrettäviksi lukuarvoiksi. Tällaisia lisälevyjä ovat muun

muassa ADS1015, joka toimii AD-muuntimena ja MAX31865, joka pystyy vahvistamaan ja muuntamaan PT100-lämpötila-antureiden signaaleita. Mittaustekniikassa yleisesti käytössä oleville analogisille 4–20 mA:n signaaleille on myös olemassa lisälevyjä signaalin muuntamiseen Raspberry Pi -tietokoneelle. Nämä kuitenkin tuovat huomattavasti lisähintaa laitteistolle, mutta antavat laitteelle rajapinnan automaation yleisimpiin antureihin. Lisälevyjä löytyy markkinoilta SPI- tai I2C-yhteyskäytännöillä, jolloin ne on mahdollista yhdistää Raspberry Pi -tietokoneelle. [43; 44; 45.]

7.4 Testausvaiheet

Työssä testattiin erilaisia vaihtoehtoja kunnonvalvonnan mittauksiin ja tiedonsiirtoon. Ensimmäisenä testattiin MQTT-yhteyskäytäntöä. Raspberry Pi -tietokoneelle asennettiin Mosquitto Broker -välittäjäohjelma, joka toimi välittäjänä KepwareServerEx-ohjelmiston ja Raspberry Pi -tietokoneen välillä. Raspberry Pi pystyi Paho-MQTT Python-ohjelmointikielisen kirjaston avulla julkaisemaan ”testi/write”-nimiseen aiheeseen (topic) lämpötila-anturin mittaamaa tietoa. Tieto saatiin siirrettyä Python-ohjelmalla lukemalla lämpötila-arvo 5 sekunnin välein ja lähettämällä se KepwareServerEx-ohjelmiston hyväksymällä string-muotoisella tietotyypillä. Lähetetyn viestin täytyi olla muotoa [{"id": "Channel1.Device1.Tag1","v": 42}], jotta KepwareServerEx-ohjelmisto pystyi tunnistamaan halutun muuttujan ja arvon. Tämä viesti lähetettiin Python-ohjelmassa Paho-mqtt-kirjaston julkaisumetodin (publish) avulla. Kanava (channel), laite (device) ja muuttuja (tag) asetettiin tähän metodiin samoiksi kuin KepwareServerEx IoT Gateway -lisäosan MQTT-asiakasohjelmassa. Mitattava lämpötila-arvo asetettiin esimerkkiviestissä luvun 42 tilalle Python-ohjelmassa. Python-ohjelma on kuvattu liitteessä 1.

KepwareServerEx IoT Gateway -lisäosaan lisättiin MQTT-asiakasohjelma, johon asetettiin Raspberry Pi -tietokoneella pyörivän MQTT-välittäjäohjelman IP-osoite ja portti. Lisäksi IoT Gatwayhin asetettiin tilaus (subscription) ”testi/write” aiheelle (topic). Tämä tilaus kuunteli aiheeseen julkaistua tietoa, jolla lämpötilatieto siirrettiin KepwareServerEx-ohjelmistolle. KepwareServerEx-ohjelmistoon asetettiin uusi kanava ”Channel2”, laite ”Device1” ja muuttuja ”tempTesti”. Nämä kytkettiin IoT Gateway -lisäosaan kirjoittamista varten. IoT Gateway MQTT client -asiakasohjelma luki Raspberry Pi -tietokoneen lähettämän lämpötilan ja kirjoitti sen Channel2.Device1.tempTesti muuttu-

jaan. Samalla KepwareServeEx kirjoittaa muuttujan arvon OPC DA -palvelimelle, josta on mahdollista siirtää nämä arvot historiatietokantaan.

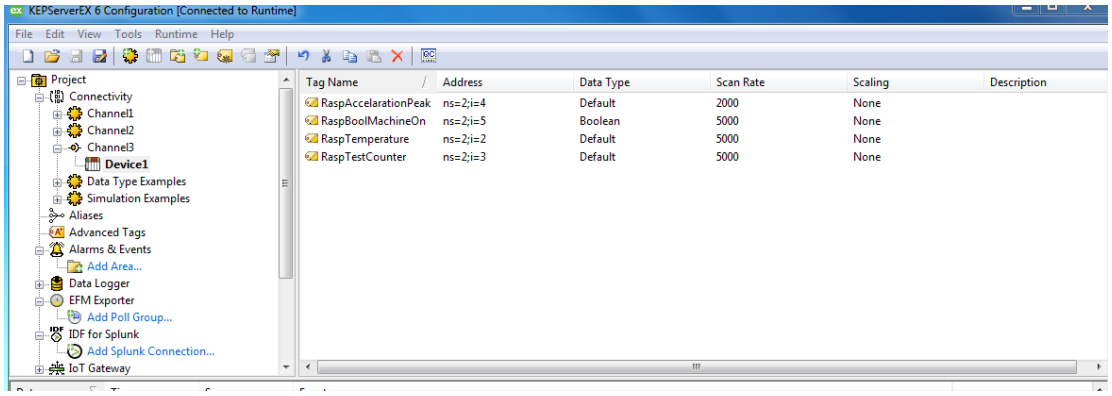
Toinen testatuista järjestelmistä oli Arduino-tietokoneelle luotu OPC DA -yhteys sekä MQTT-yhteys. Arduino-tietokoneelle ladattiin OPC DA -kirjasto testiä varten, mutta tämä vaati erillisen ohjelman, joka ladattaisiin erilliselle Windows-tietokoneelle. Tämä ei toisi hyötyä muihin järjestelmiin verrattuna, sillä niissä kaikissa tarvitaan erillinen palvelin tai tietokone tiedonsiirtoon. OPC DA on myös jo vanha standardi, joka toimii vain Windows-järjestelmissä, jolloin sen liittäminen uusiin järjestelmiin tulevaisuudessa ei ole helppoa.

Arduino-tietokoneelle testattiin myös MQTT-tiedonsiirrolla toimivaa mittausta. Arduino-tietokoneen jäykemmän käyttöliittymän vuoksi tietoverkkoyhteyksien luominen oli huomattavasti vaikeampaa kuin Raspberry Pi -tietokoneella. Tämän lisäksi Arduino-tietokoneen käyttämä oma ohjelmointikieli hankaloittaa valmiiden ohjelmien löytämistä ja joustavien ohjelmien luomista. Tästä syystä päädyttiin siihen, ettei luoda ratkaisua Arduino-tietokoneella, vaan toteutetaan laite käyttäen Raspberry Pi -tietokonetta.

Lopuksi vielä testattiin Raspberry Pi -tietokoneella luotua OPC UA -järjestelmää. Järjestelmää luotaessa mietittiin, olisiko parempaa sijoittaa OPC UA -palvelin Windows-tietokoneelle vai Raspberry Pi -tietokoneelle. Asiakasohjelma sijoitettaisiin sille tietokoneelle, jolle ei palvelinta asennettu. Työssä päädyttiin kokeilemaan molempia tapoja. Windows-tietokoneella KepwareServerEx-ohjelmisto toteuttaa OPC UA -keskustelun ja Raspberry Pi -tietokoneella se toimii FreeOPCUA-kirjaston ohjelmalla.

Windows-tietokoneen OPC UA -asiakasohjelma alustettiin luomalla uusi kanava KepwareServerEx-ohjelman asetusikkunassa (kuva 20), johon valittiin vaihtoehdoksi OPC UA Client -ajuri. Ajuriin asetettiin Raspberry Pi -tietokoneen OPC UA -palvelimen loppupiste, joka oli muotoa opcua.tcp://192.168.0.1:4840/freeopcua/server. Salausasetukset poistettiin käytöstä yhteyksien yksinkertaistamiseksi. Kanavalle asetettiin vielä uusi laite, jonka avulla asetettiin ajurille muuttujat. Muuttujille päätettiin sopivat nimet, Raspberry Pi -tietokoneelle olevan OPC UA -palvelimen mukaiset OPC UA -osoitteet, tietotyyppi ja lukunopeus. OPC UA -osoitteet muodostuivat muotoon ns=2;i=2, jossa ensimmäinen numero kertoo palvelimen nimiavaruuden (namespace) ja toinen numero avaruuden yhtymäkohdan tunnistenumeron. Nämä määritelleen palvelimen asetuksissa niin, että palvelimella on yksi nimiavaruus ja jokaisella mitattavalla muuttujalla on

oma yhtymäkohtatunnisteensa, jonka avulla asiakasohjelma löytää palvelimelta oikeat muuttujat.



Kuva 20. KepwareServerEx 6 -ohjelmiston asetusnäkymä testimuuttujille

OPC UA -palvelin luotiin Raspberry Pi -tietokoneelle asentamalla sille FreeOPCUa-Python-ohjelmointikirjasto. Tämän kirjaston avulla voitiin tehdä yksinkertainen OPC UA -palvelin, jolle muuttujien kirjoittaminen ja lukeminen onnistuivat. Kirjaston metodeiden avulla alustettiin palvelimen tiedot esimerkkikoodissa 2 ja liitteessä 2, kuten IP-osoite, objektit ja muuttujat. Tämän jälkeen palvelin voidaan käynnistää start-metodilla ja muuttujien arvon muuttaminen onnistuu käyttäen set_value-metodia. Python FreeOP-CUA -kirjaston metodit löytyvät Python-ohjelmointikirjastosta [20].

```
server = Server()
server.set_endpoint("opc.tcp://192.168.0.2:4840/freeopcua/server/")

objects = server.get_objects_node()
myobj = objects.add_object(idx, "MyObject")
myvar = myobj.add_variable(idx, "RaspTemperature", 1.0)
myvar2 = myobj.add_variable(idx, "RaspCounterTest", 1.0)
myvar3 = myobj.add_variable(idx, "RaspAccelarationPeak", 1.0)
myvar4 = myobj.add_variable(idx, "RaspBoolMachineOn", 0)

server.start()
myvar.set_value(temp)
```

Esimerkkikoodi 2. Ote OPC UA -palvelimen koodista.

Samassa ohjelmassa luettiin antureiden tiedot ja tallennettiin ne muuttujiin. Tähän käytettiin valmiita kirjastoja, joihin oli ohjelmoitu ajurit lukemaan tarvittavia antureita. Lämpötila-anturilla käytettiin SPI-yhteyskäytäntöä lukemaan analogisen anturin lähettämää tietoa. Tieto piti ensin muuntaa analogisesta digitaaliseksi AD-muuntimen avulla. Tämän jälkeen SPI-kirjaston metodeilla pystyttiin kirjoittamaan anturin antama lämpötila

sopivalla skaalauksella käyttämällä aiemmin esitettyä kaavaa 1. Tärinäanturia varten testattiin molempia I2C- ja SPI-yhteyskäytäntöjä, joissa molemmissa oli omat etunsa. I2C-yhteyskäytännöllä saadaan vähemmän häiriöitä pidemmillä johtimilla ja SPI-yhteyskäytäntö pystyy lukemaan anturin tietoja huomattavasti suuremmalla taajuudella.

SPI-yhteyskäytännön avulla pystyttiin siirtämään kiihtyvyysanturin tietoa 3200 kertaa sekunnissa käyttäen liitteessä 3 esiteltyä C-ohjelmaa. Ohjelma perustuu ADXL345spi-kirjastoon, josta työssä käytetty ohjelma on muokattu. Ohjelma lukee AD-muuntimelta saatavat digitaaliset viestit ja yhdistää ne kiihtyvyysarvoiksi. Ohjelma analysoi arvot huipusta huippuun arvoksi ja tulostaa ne OPC UA -palvelinta pyörittävälle ohjelmalle sopivalla tavalla. OPC UA -palvelimen ollessa Python-kielinen ja SPI-yhteyskäytännöllä antureita lukeva C-ohjelma täytyi näiden kahden ohjelman väliin löytää sopiva siirtotapa Raspberry Pi -tietokoneen sisäisillä toiminnoilla. Tähän käytettiin Python-kielen subprocess.check_output-metodia, jonka avulla voidaan käynnistää toinen ohjelma ja lukea sen lähettämä viesti, joka tässä tapauksessa lähetetään c-ohjelman lopussa printf-komennolla peak-muuttujan avulla.

Tärinän mittausta varten jouduttiin analysoimaan tietoa ohjelmien sisällä. Tärinää mitattiin mahdollisimman korkealla taajuudella yhden sekunnin verran. Jokaisen mittauksen jälkeen ohjelma tarkisti, onko arvo korkeampi kuin korkein arvo tai matalampi kuin matalin arvo sen hetkisen sekunnin mittaiselta ajanjaksolta. Jos arvo oli korkein tai matalin, se tallennettiin muuttujaan, jonka jälkeen sitä taas verrattiin uuteen arvoon. Kun sekunti oli kulunut, korkein ja matalin arvo vähentää toisistaan, jolloin saatiin huipusta huippuun kiihtyvyysero. Tämä arvo lähetettiin tietokannalle sekunnin välein käyttäen OPC UA -yhteyskäytäntöä.

Vaihtoehtoisesti Raspberry Pi -tietokoneella toimivaan OPC UA -palvelimeen, Raspberry Pi -tietokoneelle asennettava OPC UA -asiakasohjelma luotiin käyttäen FreeOPCUA-kirjaston asiakasohjelmaosuutta, jonka metodeilla pystyttiin luomaan yhteys KepwareServerEx-ohjelmiston OPC UA -palvelimeen. Yhteyden luomisen jälkeen get_node-metodilla haettiin KepwareServerEx OPC UA -palvelimelta halutut muuttujat palvelimelle asetetun nimiavaruuden numeron ja muuttujan numerotunnuksen avulla. Set_value-metodilla voitiin näihin haettuihin muuttujiin kirjoittaa antureiden arvot. Antureiden lukeminen asetettiin samalla tavalla kuin OPC UA -palvelinta tehtäessä.

Molemmilla vaihtoehdoilla tieto liikkui KepwareServerEx-ohjelmiston ja Raspberry Pi -tietokoneen välillä. Kuitenkin molemmissa oli hyvät ja huonot puolensa. KepwareServerEx-ohjelmiston OPC UA -palvelimessa hyötynä on se, että useammasta mittauslaitteesta voidaan kirjoittaa yhdelle palvelimelle sen sijaan, että jokaisella mittauslaitteella olisi oma palvelimensa. Tällöin voidaan muokata yhdestä paikkaa palvelimen asetuksia, mikä vähentää asetusaikaa. Kuitenkin KepwareServerEx ohjelmalla voidaan lukea hyvin myös useampaa palvelinta käyttäen eri kanavia (channel). Tällöin KepwareServerEx-ohjelmiston toimiessa asiakasohjelmana luetut tiedot ovat ajan tasalla aina ja eivät virheellisesti näytä hyvälaatuisilta. Raspberry Pi -tietokoneen OPC UA -asiakasohjelmalla kirjoittaessa ongelmana oli yhteyden katketessa se, etteivät KepwareServerEx-palvelinohjelmiston muuttujat tunnistaneet yhteyden katkeamista. Tällöin muuttujien arvot ovat kokoajan laadultaan hyviä, vaikka lukemisen tai kirjoittamisen epäonnistuessa muuttujan pitäisi saada huonolaatuinen arvo.

KepwareServerEx-ohjelmistolta tieto siirrettiin historiatietokannalle käyttäen KepwareServerEx-ohjelmiston OPC DA -palvelinta. KepwareServerEx-ohjelman ominaisuutena kaikki sille asetetut muuttujat löytyvät OPC DA -palvelimelta. Palvelinta luettiin tietokannan collector-ajuria käyttäen, johon kyettiin asettamaan käytetyn KepwareServerEx-ohjelman OPC DA -palvelin. Historiatietokannan päässä oli mahdollista määritellä, mitä muuttujia OPC DA -palvelimelta luetaan ja kuinka usein ne luetaan.

Ongelmana FreeOPCUA-kirjastojen käytössä oli, ettei niissä löytynyt metodia yhteyden uudelleen luontiin. Kun Raspberry Pi -tietokoneelle asetetun asiakasohjelman yhteys katkesi, käytetyt Python-ohjelmat kaatuivat ja lopettivat toimintansa. Kun yhteys saatiin takaisin, ei Raspberry Pi -tietokoneella enää pyörinyt OPC UA -ohjelmia. Tähän ongelmaan ratkaisuksi luotiin ohjelma, joka tarkastaa minuutin välein, ovatko OPC UA -Python-ohjelmat käynnissä. Jos ohjelmat eivät ole käynnissä, tarkastusohjelma käynnistää halutun OPC UA -Python-ohjelman uudestaan, muuten tarkastusohjelma ei tee mitään. Ohjelma asetettiin Raspberry Pi -tietokoneen crontab-ohjelmaan. Crontab on ohjelma, jolla pystytään asettamaan tietyn väliajoin ajettavia terminaalikomentoja. Tähän asetettiin uudelleenkäynnistysohjelma ja itse OPC UA -Python-ohjelma, joka asetettiin käynnistymään Raspberry Pi -tietokoneen käynnistyksen yhteydessä. Näin Raspberry Pi pystyy Crontab-ohjelman avulla käynnistämään Python-ohjelman aina kun laite saa virtaa ja uudelleenkäynnistysohjelma pitää mittaus- ja tiedonsiirto-ohjelmat käynnissä. Samaa ohjelmaa voidaan käyttää myös FreeOPCUA-palvelimen

ylläpitoon, jolloin laite pystyy automaattisesti käynnistämään palvelimen uudestaan ohjelman kaatuessa.

Toisena ongelmana huomattiin Raspberry Pi -tietokoneen kello, joka ei päivity, jollei laite ole kytkettynä internetiin. Tämä kello pysähtyy aina kun laitteesta kytketään virrat pois, joten tällöin kello on heti väärässä virtojen katkaisun jälkeen. Raspberry Pi -tietokone käyttää kellon päivittämiseen normaalisti tietoverkon välitykselle toimivaa NTP-yhteyskäytäntöä (Network Protocol Time), jolla asiakasohjelma ottaa yhteyden NTP-palvelimeen ja päivittää kellonsa tätä käyttäen [46]. Kello vaikuttaa mittaustietojen aikaleimoihin, joiden avulla mittaustiedot tallennetaan tietokantaan. Ongelmaan ratkaisuksi voidaan joko rakentaa NTP-palvelin yrityksen verkkoon, jolloin laitteen ei tarvitse kytkeytyä ulkoiseen tietoverkkoon.

Toinen vaihtoehto kellon asettamiseen on käyttää KepwareServerEx -ohjelmiston OPC UA -palvelinta ja lukea päiväys ja kellon aika tämän järjestelmätiedoista. Käyttäen Raspberry Pi -tietokoneelle asennettua OPC UA -asiakasohjelmaa voidaan KepwareServerEx-ohjelmiston OPC UA:n ”_System”-objektin aika- ja päiväysmuuttujista lukea, mittausohjelmaa alustaessa, sen hetkinen aika. Aika ja päivämäärä asetetaan Raspberry Pi -tietokoneelle ”date”-komentoa käyttäen, jonka jälkeen sijoitetaan aika järjestyksessä ”MMDDhhmmYY”. Järjestys koostuu aina kahdesta merkistä, jotka ovat järjestyksessä kuukausi, päivä, tunti, minuutti ja vuosi. Tieto muokataan luku ohjelmassa tähän muotoon, jonka jälkeen komento ajetaan ajanlukuohjelmassa.

7.5 Vaihtoehtoiset ratkaisut

Testatuissa versioissa päädyttiin ratkaisuihin, jotka mittaavat tietoa ja siirtävät sen tietokantaan tietoverkkojen yli käyttäen langallista Ethernet-yhteyttä. Normaalin Ethernet-yhteyden lisäksi Raspberry Pi -tietokoneelle on tarjolla Power over Ethernet -lisäosa [47]. Tällöin tietoverkkoyhteyden lisäksi voidaan laitteelle syöttää käyttöjännite käyttäen PoE-kaapelia, joka vaatii toimiakseen PoE-reitittimen. PoE-tekniikan käyttäminen tuo kuitenkin lisäkustannuksia erillisten reitittimien ja lisäosien hankkimiseen, mutta tällöin laitteelle ei tarvita erillistä virransyöttöä. Vaihtoehtoisesti laitetta voidaan muokata sopivaksi erilaisiin mittaustehtäviin esimerkiksi käyttäen langatonta tietoverkkoyhteyttä tai luomalla versio ilman reaaliaikaista tietoverkkoyhteyttä.

Raspberry Pi -tietokoneessa on oma WLAN-siru, jolla voidaan luoda yhteys haluttuun langattomaan tietoverkkoon. Langattomuus helpottaa laitteen asentamista mitattavan kohteen läheisyyteen, jolloin ei Ethernet-kaapelia tarvitse vetää erikseen laitteelle. Jos laitteelle lisätään erillinen akku, voi laite olla täysin langaton. Langattomuus tuo kuitenkin myös ongelmia liittyen langattoman tietoverkon kuuluvuuteen ja akun kestoon. Mittalaite joudutaan usein asentamaan metallisen sähkökaapin sisään, jotta laite on turvassa tuotantoympäristöltä. Metalliset kaapit voivat haitata langattoman tietoverkkoyhteyden kulkua, jolloin on testattava, häiritseekö mittauskohteen suojaus langatonta tietoverkkoyhteyttä.

Toinen vaihtoehto on tehdä versio laitteesta, joka pystyy tallentamaan tietoa paikallisesti. Tämä vaihtoehto voi sopia erityisesti, jos halutaan analysoida kaikkea mittaustietoa, tarvitaan korkeaa mittaustaajuutta tai tarkkaa tärinän mittausta. Tällöin Raspberry Pi voi tallentaa mittaustietoa joko sille asennettuun erilliseen tietokantaan tai tekstitiedostoihin. Tällöin tiedot voidaan ladata tietyn määräajan kuluttua pois Raspberry Pi -tietokoneelta tiedon prosessointia ja vertailua varten. Reaaliaikaisesti tuhansien mittausten lähettäminen sekunneissa monista laitteista aiheuttaa tietoverkossa katkoksia ja hidastaa tietoverkon toimintaa, jolloin ei pystytä lähettämään reaaliaikaista tärinämittausta tietoverkossa [48]. Tällöin tietoa pitää analysoida mittauslaitteella tai se voidaan siirtää isompina paketteina jälkikäteen. Menetelmän haittapuolina ovat reaaliaikaisen tiedon menettäminen ja jälkikäteen tiedonsiirtämisen aiheuttama lisätyö ohjelmoinnissa tai käsin siirtämisessä. Lämpötilaa mitattaessa haluttaisiin, että mittaus tehdään reaaliaikaisesti, sillä lämpötilan kasvaessa laite voi olla jo hyvin lähellä hajoamispistettä [2, s. 5].

Kolmas vaihtoehto on käyttää laitteen Bluetooth-ominaisuutta, jonka avulla voidaan kytkeä langattomia antureita Raspberry Pi -tietokoneeseen. Tällöin Raspberry Pi toimii pelkästään tiedon analysoijana ja siirtäjänä, mikä helpottaa laitteen asentamista mitattavien laitteistojen läheisyyteen. Langaton yhteys tuo kuitenkin mukanansa mahdolliset yhteyshäiriöt ja kasvattaa laitekustannuksia Bluetooth-antureiden tarvitessa omaa älyä tiedonsiirtoa varten. Bluetooth-ominaisuutta varten on myös rakennettava oma ohjelma, joka pystyisi siirtämään Bluetooth-tiedon OPC UA -ohjelmalle luettavaan muotoon.

8 Lopputulos

Työssä luotiin esittelyversio laitteesta, jolla pystytään mittamaan lämpötilaa, tärinää ja käyntitietoa käyttäen Raspberry Pi -tietokonetta. Raspberry Pi siirtää mitatut arvot tietokantaan lähettämällä ne ensin KepwareServerEx-ohjelmistoa pyörittävälle palvelimelle tai PC:lle käyttäen OPC UA -yhteyskäytäntöä. KepwareServerEx-ohjelmisto pystyy lukemaan OPC UA -palvelimen lähettämää tietoa ja kykenee siirtämään tiedon myös eteenpäin tietokannalle. Työssä testattuun tietokantaan tarvittiin OPC DA -palvelin, jona KepwareServerEx-ohjelma toimi.

MQTT-järjestelmää ei yrityksen järjestelmistä johtuen voida ottaa käyttöön ainakaan vielä. Tulevaisuudessa kuitenkin on mahdollista, esimerkiksi käyttäen KepwareServerEx-ohjelmiston IoT Gateway -lisäosaa, luoda yhteydet myös MQTT-yhteyskäytännöllä. Tämä vähentäisi tietoliikenteen määrä sekä keventäisi tarvittavia järjestelmiä.

Raspberry Pi -tietokone valikoitui käyttöalustaksi sen

- kaupallisen saatavuuden
- edullisuuden
- muokattavuuden
- riittävän tehon
- digitaalisten sisääntulojen
- tietoverkkoyhteyden
- älykkään mittauksen mahdollisuuden
- siirrettävyyden takia.

Nämä ominaisuudet vastasivat yrityksen toiveita niiltä määrin, kun laitetta käytettäisiin kunnonvalvonnan apuna mittaamaan laitteita, joille ei ole olemassa valmista kunnonvalvontajärjestelmää. Lisäksi sopivilla lisälevyillä pystytään muuttamaan haluttuja analogisia mittaustietoja mittalaitteelle sopiviksi digitaalisiksi tiedoiksi, jolloin voidaan laitteen mittauksiin lisätä muiltakin anturityypeiltä saatavia analogisignaaleita. Kunnonvalvontalaitteelle valittiin tiedonsiirtoon OPC UA -yhteyskäytäntö, koska se on yleisesti käytössä useissa automaation järjestelmissä ja insinöörityön tilanneessa yrityksessä. KepwareServerEX-ohjelmisto on myös yritykselle tuttu, mikä vähentää uuden osaamisen tarvetta. Tämä ohjelmisto on rakennettu OPC-viestinnän pohjalta, jolloin se sopii sillaksi mittauslaitteen ja tietokannan väliseen OPC UA- ja OPC DA -viestintään.

Laitteistoa voidaan käyttää kunnonvalvonnan apuna yrityksessä, sillä laitteisto ei tarvitse virtalähteen lisäksi muuta kuin tietoverkkoyhteyden, joka laitteesta löytyy niin langattomana kuin langallisena. Tällöin laite voidaan kiinnittää tehtaassa monen mittatavan laitteiston yhteyteen, joista voidaan mitata esimerkiksi moottorin tai puhaltimen kuntoa. Jos mittaustietoa pystytään tulkitsemaan oikein, voidaan kunnonvalvontalaitteiston avulla huomata alkavia vikoja mekaanisten ja sähköisten laitteiden toiminnassa.

Esittelylaitteiston tärinämittauksessa on kuitenkin rajoitteita, jotka voivat haitata tarkan tiedon saamista. Laitteen mittaustaajuus oli vain noin 700 hertsiä I2C-yhteyskäytännöllä, jolloin korkeammat tärinän taajuudet voivat jäädä huomaamatta. SPI-yhteyskäytäntöä käytettäessä tärinän mittaukseen täytyy ottaa huomioon, kuinka pitkillä johtimilla anturia voidaan käyttää. Tällä käytännöllä kuitenkin saadaan mittaustaajuudeksi 3200 hertsiä. Korkeataajuisten tärinöiden mittaukseen laite ei kykene, joten laitteella ei voida analysoida tärinätaajuuksia tarkemmin. Jos laitteesta aiotaan tehdä konkreettinen ratkaisu, joudutaan miettimään antureiden taajuuden tarkkuutta käyttökohteesta riippuen. Työn tarkoituksena ei kuitenkaan ollut tarkastella anturitekniikkaa, vaan luoda mahdollinen ratkaisu älykkääseen tiedonsiirtoon antureiden ja tietokannan välillä. Kehitys ideana tärinän mittaukseen olisi myös kiihtyvyys arvon muuntaminen nopeudeksi esimerkiksi Fourier-muunnosta käyttäen, milloin se vastaa paremmin nykyaikaisia standardeja.

Tulevaisuudessa laitteistoa voidaan kehittää hankkimalla uusia antureita, jotka mittaavat eri asioita tai ovat tarkempia. Esimerkiksi automaation yleisiä 4–20 mA:n virtaviestejä voidaan hyödyntää Raspberry Pi -tietokoneelle luotujen lisälevyjen avulla. OPC UA -yhteyskäytäntö todennäköisesti tulee olemaan yhä hallitseva yhteyskäytäntö tiedonsiirtoon automaation saralla, joten työssä esiteltyjä tapoja voidaan hyödyntää myös tulevaisuuden järjestelmissä ja laitteistoissa. Laitteiston joustavuuden vuoksi on myös mahdollista tehdä uudenlaisia viestintä ja mittausjärjestelmiä tarvittaessa, jos resursseja laitteiston kehitykseen löytyy. Laitteisto vaatii kuitenkin resursseja ylläpitoa ja laitteen monistamista varten, jolloin yrityksessä vaaditaan osaamista laitteen toiminnasta tai se joudutaan hankkimaan ulkoisilta toimijoilta. Työssä hankittua osaamista on kuitenkin mahdollista käyttää myös muiden vastaavien järjestelmien kanssa, esimerkiksi niiden tiedonsiirtojärjestelmiä suunnitellessa. Laitteistoa on myös mahdollista muokata muihin tiedonkeruutehtäviin, joissa tarvittaisiin halpaa reaaliaikaista suureiden mittausta.

Lähteet

1 PSK6201 Kunnossapito, Käsitteet ja Määritelmät. 2003. Standardi.

2 ABB. 2000. TTT-käsikirja. Verkkoaineisto. <http://www.oamk.fi/~kurki/automaatiolabrat/TTT/23_Kunnonvalvonta%20ja%20huolto.pdf>. Luettu 25.1.2018.

3 Mobley, R. K. 2002. An Introduction to Predictive Maintenance. USA: Butterworth Heinemann.

4 Asp, Risto & Tuominen, Timo & Hyppönen, Heikki. Verkkoaineisto. <http://www03.edu.fi/oppimateriaalit/kunnossapito/perusteet_2-1_kunnossapidon_kasitteet_ja_maaritelmat.html>. Luettu 25.1.2018

5 SFS-ISO 13373 Koneiden kunnonvalvonta ja diagnostiikka. Värähtelytilan valvonta. 2012. Standardi. Suomen Standardisoimisliitto SFS.

6 Understanding the benefits of vibration monitoring and analysis. Verkkoaineisto. Fluke. <http://en-us.fluke.com/community/fluke-news-plus/vibration/understanding-the-benefits-of-vibration-monitoring-and-analysis.html>. Luettu 25.1.2018.

7 SFS-EN 13306 Maintenance terminology. 2017. Metalliteollisuuden Standardisointiyhdistys ry. Standardi.

8 Moxa ioLogik2240 spefications. Verkkoaineisto. <https://www.moxa.com/product/ioLogik-E2240.htm>. Luettu 15.2.2018.

9 Frequently asked questions. Verkkoaineisto. <http://mqtt.org/faq>. Luettu 25.1.2018.

10 About Oasis. Verkkoaineisto. Oasis. <https://www.oasis-open.org/org>. Luettu 29.1.2018.

11 TCP/IP-protocol Architecture. Verkkoaineisto. Microsoft. <https://technet.microsoft.com/en-us/library/cc958821.aspx> Luettu 29.1.2018.

12 Oasis standards. Verkkoaineisto. <https://www.oasis-open.org/standards#mqttv3.1.1>. Luettu 25.1.2018.

13 MQTT protocol messages overview. Verkkoaineisto. <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>. Luettu 25.1.2018.

14 Mosquitto. Verkkoaineisto. <https://mosquitto.org/>. Luettu 29.1.2018

15 Mosquitto configuration. Verkkoaineisto. <https://mosquitto.org/man/mosquitto-conf-5.html>. Luettu 29.1.2018.

16 Paho-MQTT. Ohjelmointikirjasto. <https://pypi.python.org/pypi/paho-mqtt/1.3.1>. Luettu 29.1.2018.

17 IoT Gateway Manual. Verkkoaineisto. Kepware. <https://www.kepware.com/en-us/products/kepserverex/advanced-plug-ins/iot-gateway/documents/iot-gateway-manual.pdf>. Luettu 29.1.2018.

18 OPC UA. Verkkoaineisto. OPC Foundation. <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Luettu 29.1.2018.

19 Mahnke, W. & Leitner, S-H. & Damm, M. 2009. OPC Unified Architecture. Berlin: Springer.

20 Python-OPCUA. Ohjelmointikirjasto. <https://github.com/FreeOpcUa/python-opcua>. Luettu 29.1.2018.

21 KepwareServerEx. Verkkoaineisto. <https://www.kepware.com/en-us/products/kepserverex/>. Luettu 29.1.2018.

22 Historian Getting Started Manual. Verkkoaineisto. GE. <https://digitalsupport.ge.com/servlet/fileField?id=0BE1A000000L4Yg>. Luettu 29.1.2018.

23 A Relational Database Overview. Verkkoaineisto. Oracle. <https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>. Luettu 29.1.2018.

24 Raspberry Pi FAQ. Verkkoaineisto. Raspberry Pi. <https://www.raspberrypi.org/help/faqs/>. Luettu 29.1.2018.

25 Raspberry Pi model 3B Product Overview. Verkkoaineisto. Raspberry Pi. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Luettu 29.1.2018.

26 SSH Secure Shell. Verkkoaineisto. Raspberry Pi. <https://www.raspberrypi.org/documentation/remote-access/ssh/>. Luettu 29.1.2018.

27 Raspberry Pi 3B. Kuva. Raspberry Pi. <https://www.raspberrypi.org/app/uploads/2017/05/Raspberry-Pi-3-1-1619x1080.jpg>. Katsottu 29.1.2018.

28 SPI documentation. Verkkoaineisto. Raspberry Pi. <https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md>. Luettu 29.1.2018.

29 How I2C communication works. Verkkoaineisto. <http://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/>. Luettu 29.1.2018.

30 Datasheet ADXL345 accelerometer. Verkkoaineisto. <https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf>. Luettu 29.1.2018.

31 Serial Peripheral Interface Tutorial. Verkkoaineisto. <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>. Luettu 29.1.2018.

32 Arduino Yun Overview. Verkkoaineisto. Arduino. <https://store.arduino.cc/arduino-yun>. Luettu 29.1.2018.

33 Arduino Yun. Kuva. Arduino. <https://store-cdn.arduino.cc/usa/catalog/product/cache/1/image/500x375/f8876a31b63532bbba4e781c30024a0a/a/0/a000008_front.jpg>. Katsottu 29.1.2018.

34 ADXL345-python. Ohjelmointikirjasto. <https://github.com/pimoroni/adxl345-python>. Luettu 29.1.2018.

35 ADXL345spi. Ohjelmointikirjasto. <https://github.com/nagimov/adxl345spi>. Luettu 29.1.2018.

36 Spidev. Ohjelmointikirjasto. <https://pypi.python.org/pypi/spidev>. Luettu 29.1.2018.

37 Linkerkit-Base-RB2. Kuva. <http://www.linkerkit.de/images/thumb/d/d6/LK-Base-RB2.jpg/358px-LK-Base-RB2.jpg>. Katsottu 29.1.2018.

38 LK-Temp-lämpötila-anturi. Kuva. <https://www.conrad.com/medias/global/ce/0000_0999/0700/0760/0767/1267861_RB_00_FB.EPS_1000.jpg>. Katsottu 29.1.2018.

39 Linkerkit LK-temp Application Ideas. Verkkoaineisto. Linkerkit. <http://www.linkerkit.de/images/d/d2/001267861-da-01-en-LINKER_KIT_PLATINE_MIT_TEMPARATURSENSOR.pdf>. Luettu 29.1.2018.

40 ADXL345-kiihtyvyysanturi. Kuva. <http://www.robotpark.com/image/cache/data/PRO/91450/91450-GY-291-ADXL345-3-Axis-Accelerometer-700x700.jpg>. Katsottu 29.1.2018.

41 Raspberry Pi GPIO pull up and pull down resistors. Verkkoaineisto. <http://www.kalitut.com/2017/11/RaspberryPi-GPIO-pull-up-pull-down-resistor.html>. Luettu 29.1.2018.

42 GPIO documentation. Verkkoaineisto. Raspberry Pi. <https://www.raspberrypi.org/documentation/usage/gpio/>. Luettu 29.1.2018.

43 ADS1015 and ADS1115. Verkkoaineisto. Adafruit. <https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115>. Luettu 29.1.2018.

44 PT100 RTD Temperature sensor apmplifier MAX31865. Verkkoaineisto. Adafruit. <https://blog.adafruit.com/2016/11/15/new-product-adafruit-pt100-rtd-temperature-sensor-amplifier-max31865/>. Luettu 29.1.2018.

45 4-20mA Current Loop Receiver mikroBUS Module. Verkkoaineisto. <http://microcontrollershop.com/product_info.php?products_id=5460>. Luettu 29.1.2018.

46 Network Time Protocol. Verkkoaineisto. Cisco. <https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-58/154-ntp.html>. Luettu 2.3.2018.

47 Raspberry Pi Poe Switch Hat. Verkkoaineisto. <https://www.pi-supply.com/product/pi-poe-switch-hat-power-over-ethernet-for-raspberry-pi>. Luettu 29.1.2018.

48 Network Congestion Definition. 2005. Verkkoaineisto. Linux Information Project. <http://www.linfo.org/congestion.html>. Luettu 29.1.2018.

OPC UA -palvelimen Python-ohjelma

```
import sys
sys.path.insert(0, "..")
import time
import spidev
import random
import RPi.GPIO as GPIO
from adxl345 import ADXL345
from opcua import ua, Server

# Setup Digital Input 26 to up mode, where it reads True when it’s grounded
GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#Setup for SPI protocol for reading analog input from analog to digital con-
verter
spi = spidev.SpiDev()
spi.open(0,0)
def readadc(adcnum):
 if adcnum >3 or adcnum <0:
  return-1
 r = spi.xfer2([1,8+adcnum <<4,0])
 adcout = ((r[1] &3) <<8)+r[2]
 return adcout


if __name__ == "__main__":

    # setup our server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.0.2:4840/freeopcua/server/")

    # setup our own namespace, not really necessary but should as spec
    uri = "http://test"
    idx = server.register_namespace(uri)

    # get Objects node, this is where we should put our nodes
    objects = server.get_objects_node()

    # populating our address space
    myobj = objects.add_object(idx, "MyObject")
    myvar = myobj.add_variable(idx, "RaspTemperature", 1.0)
    myvar2 = myobj.add_variable(idx, "RaspCounterTest", 1.0)
    myvar3 = myobj.add_variable(idx, "RaspAccelarationPeak", 1.0)
    myvar4 = myobj.add_variable(idx, "RaspBoolMachineOn", 0)


    # starting!
    server.start()

    try:
        #Setup accelerometer from adxl345 driver library
        adxl345 = ADXL345()
        #setup variables i and j for loops
        i=0
        j=0
        #setup variable arvolist to capture a list of accelerometer data
        arvolist = []
        #setup Accelerometer bandwidth to 1600Hz and range to 4G
```

```
    adxl345.setBandwidthRate(0x0F)
    adxl345.setRange(0x01)
    #setup variable for random number for testing purposes
    count = 1

    while True:
        #Start reading axes
        axes = adxl345.getAxes(True)
        #variables for storing Axes acceleration, for now only z needed
        #x=  axes['x']
        #y=  axes['y']
        z =  axes['z']
        #add z value to arvolist list of acceleration data
        arvolist.append(z)
        i+=1
        j+=1

        if j>5000:
           #Temperature value read with SPI from analog input 0
           value=readadc(0)
           #Change digital bit data to float temperature
           volts=(value*3.3)/1024
           temp=(volts-0.5)*100
           #random number for test purposes
           count=random.randint(1, 20)

           #Read Digital Input 26
           input_state = GPIO.input(26)

           #Invert data because input_state is set in up mode
           #in up mode when input is grounded it will show False
           #it's normally True, when ground and input are not connected
           if input_state == False:
              MachineOn=True
              myvar4.set_value(MachineOn)
           else:
              MachineOn=False
              myvar4.set_value(MachineOn)
           #setting variable values for OPC UA Server
           myvar2.set_value(count)
           myvar.set_value(temp)
           j = 0

        if i>1400:
           #Check list sized of i, which value is the largest and smallest
           maxarvo=max(arvolist)
           minarvo=min(arvolist)
           #Calculating value peak to peak from highest to lowest
           peaktopeak=maxarvo-minarvo
           myvar3.set_value(peaktopeak)
           #emptying list for next round of data
           del arvolist[:]
           i=0
finally:
    #close connection, remove subcsriptions, etc
    server.stop()
```

MQTT-tiedonsiirtoon Python-ohjelma

```
import spidev
import time
import paho.mqtt.client as paho

#MQTT Client Setup
client = paho.Client()
client.connect("127.0.0.1")
client.loop_start()

#SPI setup
spi = spidev.SpiDev()
spi.open(0,0)

#Read ADC / Temperature analog -> digital
def readadc(adcnum):
 if adcnum >3 or adcnum <0:
  return-1
 r = spi.xfer2([1,8+adcnum <<4,0])
 adcout = ((r[1] &3) <<8)+r[2]
 return adcout

while True:
 #transform digital to temperature value in integer
 value=readadc(0)
 volts=(value*3.3)/1024
 temp=(volts-0.5)*100
 temp2=int(temp)
 #publish to Kepware iotgateway/write topic
 client.publish("iotgateway/write", "[{\"id\":
\"Channel1.Device1.test\",\"v\": %d}]"%(temp2))
 time.sleep(2.0)
```

Kiihtyvyysanturin ADXL345 C-ohjelma SPI-yhteyskäytännöllä lukemiseen

```
#include <stdio.h>
#include <pigpio.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define DATA_FORMAT   0x31  // data format register address
#define DATA_FORMAT_B 0x0B  // data format bytes: +/- 16g range, 13-bit reso-
lution (p. 26 of ADXL345 datasheet)
#define READ_BIT      0x80
#define MULTI_BIT     0x40
#define BW_RATE       0x2C
#define POWER_CTL     0x2D
#define DATAX0        0x32

//const char codeVersion[3] = "0.2";  // code version number
const int timeDefault = 5;  // default duration of data stream, seconds
const int freqDefault = 5;  // default sampling rate of data stream, Hz
const int freqMax = 3200;  // maximal allowed cmdline arg sampling rate of
data stream, Hz
const int speedSPI = 2000000;  // SPI communication speed, bps
const int freqMaxSPI = 100000;  // maximal possible communication sampling
rate through SPI, Hz (assumption)
const int coldStartSamples = 2;  // number of samples to be read before out-
putting data to console (cold start delays)
const double coldStartDelay = 0.1;  // time delay between cold start reads
const double accConversion = 2 * 16.0 / 8192.0;  // +/- 16g range, 13-bit res-
olution
//const double tStatusReport = 1;  // time period of status report if data
read to file, seconds

int readBytes(int handle, char *data, int count) {
    data[0] |= READ_BIT;
    if (count > 1) data[0] |= MULTI_BIT;
    return spiXfer(handle, data, data, count);
}

int writeBytes(int handle, char *data, int count) {
    if (count > 1) data[0] |= MULTI_BIT;
    return spiWrite(handle, data, count);
}

int main(int argc, char *argv[]) {
    int i;

    // handling command-line arguments


    double vTime = timeDefault;
    double vFreq = timeDefault;
    for (i = 1; i < argc; i++) {  // skip argv[0] (program name)
        if ((strcmp(argv[i], "-t") == 0) || (strcmp(argv[i], "--time") == 0))
{
            if (i + 1 <= argc - 1) {
                i++;
                vTime = atoi(argv[i]);
            }
```

```
            else {

                return 1;
            }
        }
        else if ((strcmp(argv[i], "-f") == 0) || (strcmp(argv[i], "--freq") ==
0)) {
            if (i + 1 <= argc - 1) {
                i++;
                vFreq = atoi(argv[i]);
                if ((vFreq < 1) || (vFreq > 3200)) {
                    printf("Wrong sampling rate specified!\n\n");

                    return 1;
                }
            }
            else {

                return 1;
            }
        }
        else {

            return 1;
        }
    }

    // reading sensor data

    // SPI sensor setup
    int samples = vFreq * vTime;
    int success = 1;
    int h, bytes;
    char data[7];
    int16_t z, max=-10000, min=10000;//, y, x;
    double zmax, zmin, peak; //tDuration, t;
    if (gpioInitialise() < 0) {
        printf("Failed to initialize GPIO!");
        return 1;
    }
    h = spiOpen(0, speedSPI, 3);
    data[0] = BW_RATE;
    data[1] = 0x0F;
    writeBytes(h, data, 2);
    data[0] = DATA_FORMAT;
    data[1] = DATA_FORMAT_B;
    writeBytes(h, data, 2);
    data[0] = POWER_CTL;
    data[1] = 0x08;
    writeBytes(h, data, 2);

    double delay = 1.0 / vFreq;  // delay between reads in seconds

    // depending from the output mode (print to cmdline / save to file) data
is read in different ways

    // for cmdline output, data is read directly to the screen with a sampling
rate which is *approximately* equal...
    // but always less than the specified value, since reading takes some time



        // fake reads to eliminate cold start timing issues (~0.01 s shift of
sampling time after the first reading)
    for (i = 0; i < coldStartSamples; i++) {
```

```
        data[0] = DATAX0;
        bytes = readBytes(h, data, 7);
        if (bytes != 7) {
            success = 0;
        }
        time_sleep(coldStartDelay);
    }
    // real reads happen here

    for (i = 0; i < samples; i++) {
        data[0] = DATAX0;
        bytes = readBytes(h, data, 7);
        if (bytes == 7) {
            //x = (data[2]<<8)|data[1];
            //y = (data[4]<<8)|data[3];
            z = (data[6]<<8)|data[5];
        }
        else {
            success = 0;
        }
        if (z>max) {
            max=z;
        }
        if (z<min) {
            min=z;

        }
        time_sleep(delay);  // pigpio sleep is accurate enough for console
output, not necessary to use nanosleep

    }
    zmax = max * accConversion;
    zmin = min * accConversion;
    peak = zmax - zmin;
    printf("%.5f\n", peak);
    gpioTerminate();

    if (success == 0) {
        printf("Error occurred!");
        return 1;
    }
    return 0;
```